Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Redes Convolucionales 2D para clasificación de gestos de manos

Autor: Vanessa Gaete
Profesor: Javier Ruiz Del Solar
Profesor Auxiliar: Patricio Loncomilla Z.
Fecha de entrega: 16 de julio de 2022

Índice de Contenidos

1. Introducción **1**

2. Gráficos de las señales y carga de los datos. **2**

3. Generación de imágenes a partir de ventanas **8**

4. Estructura de red utilizada **9**

5. Análisis preliminar eliminación y filtrado de imágenes **10**

6. Resultados **12**
6.1. Conv2D con espectrogramas 12
6.2. Conv2D con señales 14
6.3. Dense con espectrogramas 15
6.4. Dense con señales 17

7. Resultados en Kaggle **19**

8. Análisis de resultados **19**

9. Problemas y posibles mejoras **21**

10. Conclusiones **22**

Índice de Figuras

1. Señales para los 8 canales de un sujeto del dataset. 2
2. Señales del canal 1 para el sujeto 36 del conjunto de entrenamiento 3
3. Señales del canal 2 para el sujeto 36 del conjunto de entrenamiento 3
4. Señales del canal 3 para el sujeto 36 del conjunto de entrenamiento 4
5. Señales del canal 2 para el sujeto 36 del conjunto de entrenamiento 4
6. Señales del canal 4 para el sujeto 36 del conjunto de entrenamiento 5
7. Señales del canal 5 para el sujeto 36 del conjunto de entrenamiento 5
8. Señales del canal 7 para el sujeto 36 del conjunto de entrenamiento 6
9. Señales del canal 8 para el sujeto 36 del conjunto de entrenamiento 6
10. Estructura de red utilizada 9
11. Public score de Kaggle sobre los modelos entrenados con eliminación de señales 10
12. Precisión del conjunto de entrenamiento y validación durante entrenamiento para modelo sin señal 1 11
13. Pérdida del conjunto de entrenamiento y validación durante entrenamiento para modelo sin señal 1 11
14. Matriz de confusión para la red sobre el conjunto de validación 12
15. Precisión de la red 1 durante el entrenamiento 13
16. Pérdida de la red 1 durante el entrenamiento 13

17. Matriz de confusión para la red sobre el conjunto de validación 14
18. Precisión de la red 2 durante el entrenamiento 14
19. Pérdida de la red 2 durante el entrenamiento 15
20. Matriz de confusión para la red sobre el conjunto de validación 15
21. Precisión de la red 3 durante el entrenamiento 16
22. Pérdida de la red 3 durante el entrenamiento 16
23. Matriz de confusión para la red sobre el conjunto de validación 17
24. Precisión de la red 4 durante el entrenamiento 17
25. Pérdida de la red 4 durante el entrenamiento 18
26. Resultados de Kaggle ordenados por score obtenido 19

Índice de Códigos

.1. Insert code directly in your document 23

1. Introducción

En el proyecto, se busca crear una red neuronal o clasificador capaz de clasificar o identificar un gesto de mano con una precisión adecuada, lo que será definido con respecto a los resultados generales de las redes o clasificadores de todo el curso. Los datos a utilizar son los del dataset EMG Data for Gestures , que consiste en la medición de gestos a 36 sujetos distintos, las mediciones consisten en 8 señales obtenidas a través de electromiogramas colocados en los antebrazos de las personas. Estas mediciones tienen una etiqueta que indica el gesto que se está haciendo en ese momento para cada instante de tiempo. Además cada sujeto tiene dos mediciones o capturas. Se decidió resolver el problema utilizando Redes Neuronales, esta decisión se tomó pensando en el aprendizaje, pues se quiere poner en práctica este sistema de clasificación que no se logró evaluar en el resto del curso.

En el presente informe se presentarán las distintas opciones que se probaron para resolver el problema y la que finalmente generó los mejores resultados de todas ellas. Así los objetivos son cargar los datos de entrenamiento+validación y los de prueba, generar los conjuntos de entrenamiento y validación, crear un sistema que obtenga ventanas de cierto tamaño y con cierto paso a partir de los datos, estudiar qué tipo de entrada a la red genera mejores resultados, estudiar la importancia de cada señal en el problema, evaluar el uso de distintas estructuras de red y mostrar los resultados obtenidos.

El proyecto se realizará en Google Colab, utilizando Python y librerías como Tensorflow.

2. Gráficos de las señales y carga de los datos.

Para cargar los datos se subieron las carpetas Dataset señales para entrenamiento y Dataset ventanas de pruebas que están en material docente a Google Drive, de esta forma se puede acceder al contenido de las carpetas en el proyecto de Google Colab.

Lo primero que se hace con el conjunto de datos es normalizar los datos por canal. Se intentaron dos formas de hacerlo, normalizar todos los datos en conjunto de un mismo canal y por otro lado, normalizar los datos de una ventana por canal. La forma que se utilizó para resolver el problema fue la primera, pues normalizando los datos de una ventana no se pudo obtener más de un 50 % de accuracy, mientras que con la segunda forma se sobrepasaba el 70 %.

Para tener una idea del problema que se esta abordando, se grafican las 8 señales para un sujeto de ejemplo, sacado del conjunto de entrenamiento y validación. Se meustra un grafico que contiene las 8 señales en un mismo gráfico y luego también por separado.

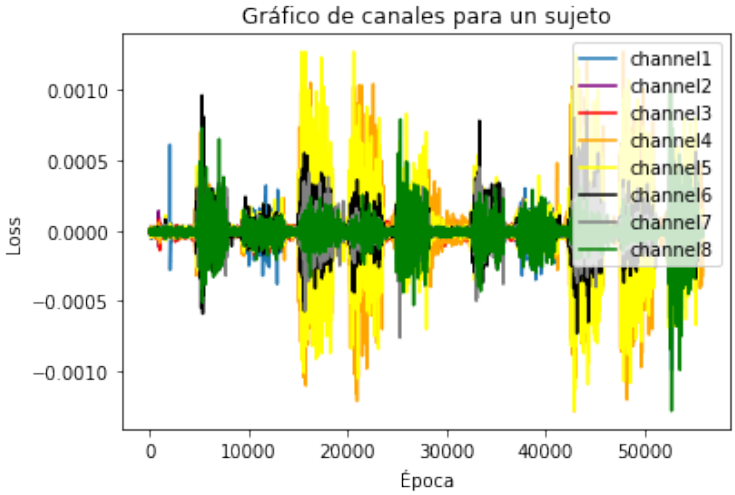

Figura 1: Señales para los 8 canales de un sujeto del dataset.

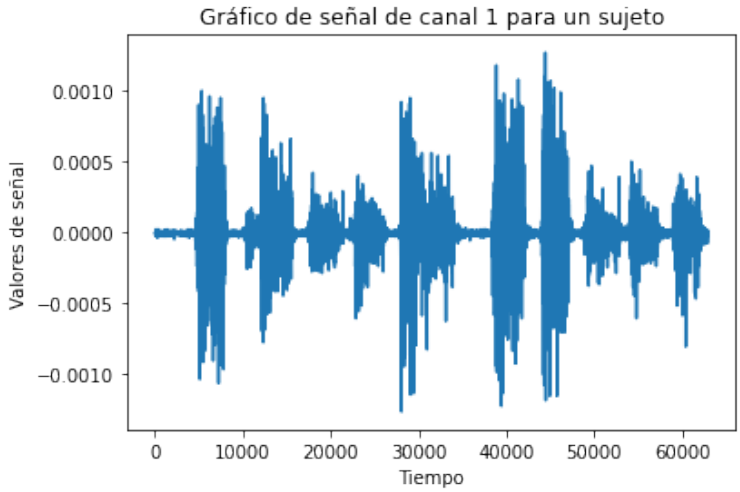

Figura 2: Señales del canal 1 para el sujeto 36 del conjunto de entrenamiento

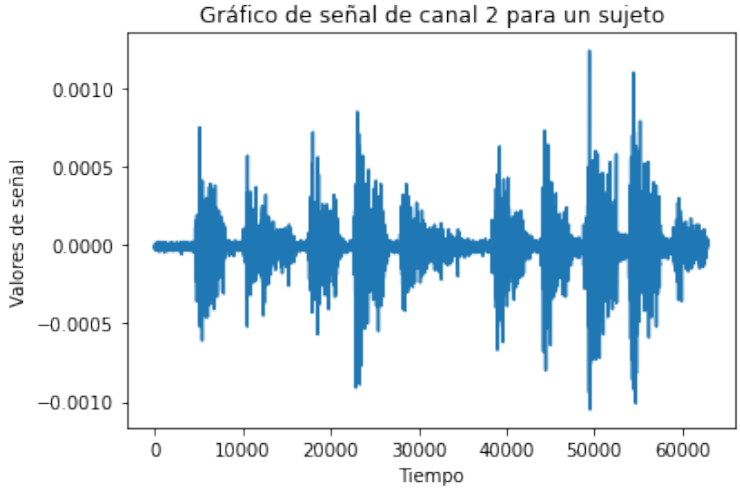

Figura 3: Señales del canal 2 para el sujeto 36 del conjunto de entrenamiento

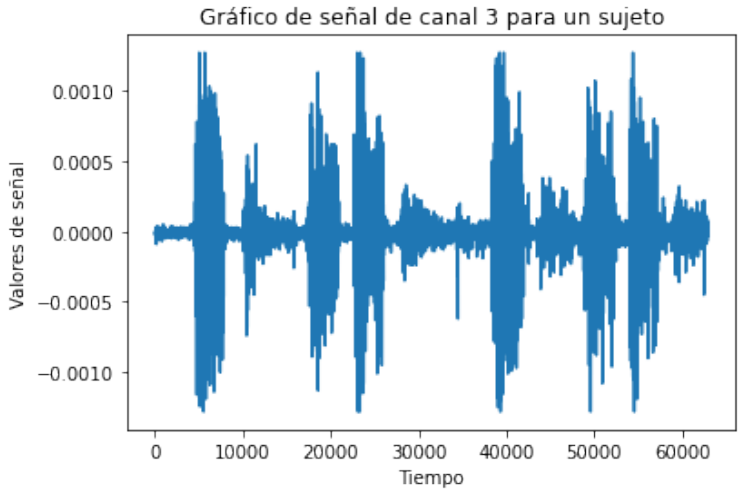

Figura 4: Señales del canal 3 para el sujeto 36 del conjunto de entrenamiento

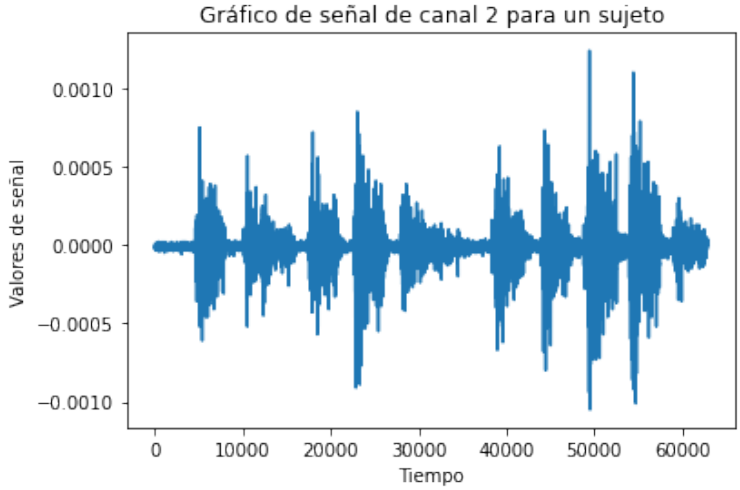

Figura 5: Señales del canal 2 para el sujeto 36 del conjunto de entrenamiento

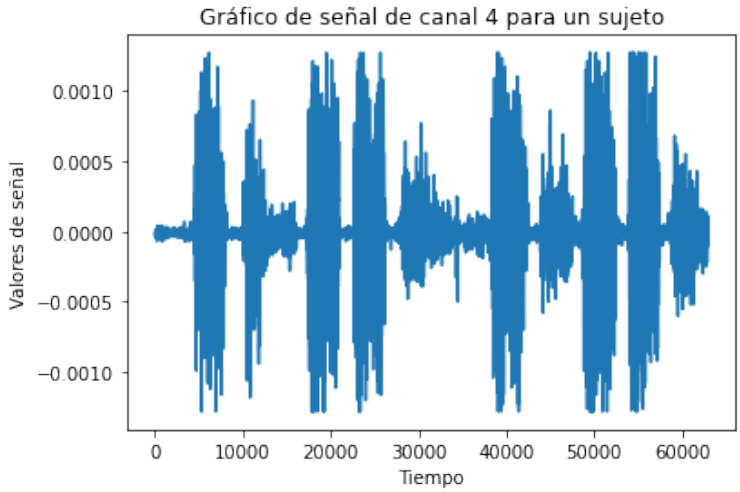

Figura 6: Señales del canal 4 para el sujeto 36 del conjunto de entrenamiento

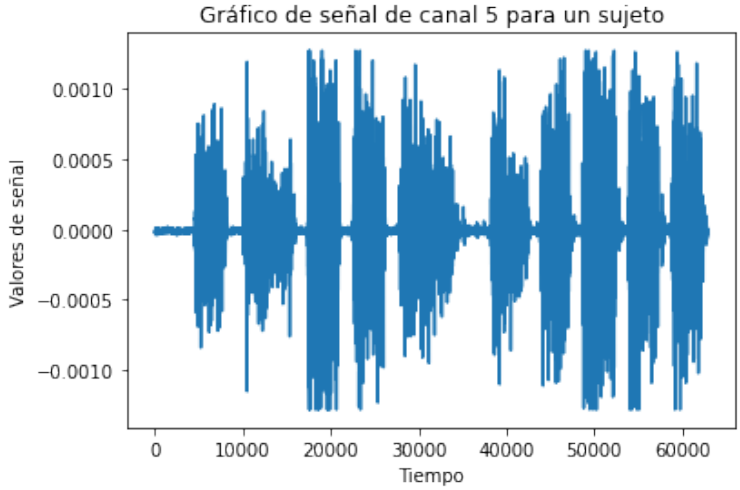

Figura 7: Señales del canal 5 para el sujeto 36 del conjunto de entrenamiento

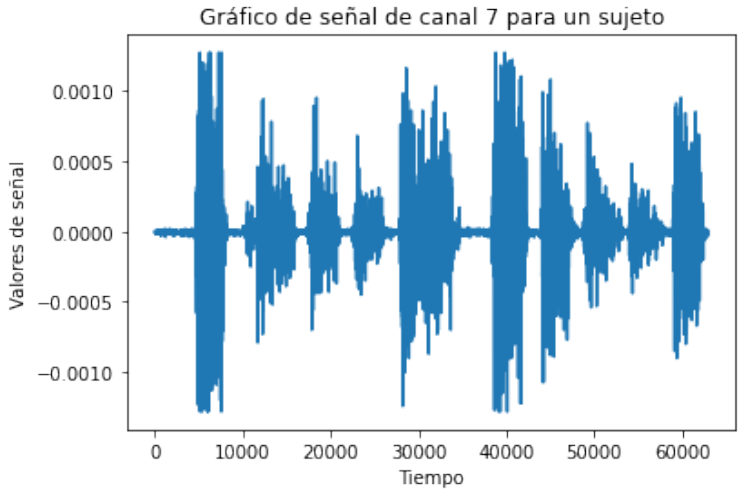

Figura 8: Señales del canal 7 para el sujeto 36 del conjunto de entrenamiento

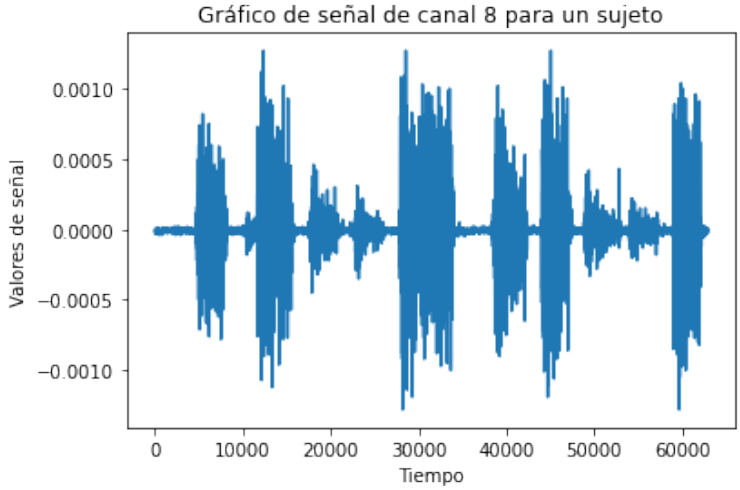

Figura 9: Señales del canal 8 para el sujeto 36 del conjunto de entrenamiento

Luego se procede a leer los datos de entrenamiento+validación para juntarlos en un único dataframe, esto se logra concatenando los datos de cada persona aumentando cada vez más la cantidad de filas del dataset. Sin embargo, no se van añadiendo todos los datos al mismo dataset, sino que antes las personas se van dividiendo entre entrenamiento y validación, y luego sus datos se adjudican al conjunto correspondiente. La razón para conformar los conjuntos es de 80 % para el conjunto de entrenamiento y 20 % para el de validación aproximadamente. Con lo anterior la estructura de los datos resulta en una matriz 2D con dimensiones (cantidad de personas x cantidad de intervalos de tiempo) x cantidad de canales.

Para resolver el problema es necesario subdividir las capturas de cada persona, pues cada captura tiene más de una etiqueta, dado que estas se asocian a intervalos de tiempo y cada captura tiene aproximadamente 50.000 intervalos. Para usar redes neuronales se necesita que cada dato que se

quiere predecir tenga tan solo un label, es por esto que se debe subdividir la captura para generar ventanas. Estas ventanas deben cumplir con la condición de pertenecer al mismo sujeto y que además todos sus intervalos de tiempo posean la misma etiqueta. El tamaño de la ventana y el paso con el que se crean (siendo el paso la cantidad de intervalos de tiempo que se mueve la ventana para generar otra) es objeto de estudio de este informe.

Hay que notar, que considerar ventanas cambia totalmente la estructura de los datos, puesto que antes eran matrices 2D y ahora pasan a ser matrices de 3 dimensiones, correspondiente a cantidad de ventanas x tamaño de ventana x cantidad de canales. Esto se aplica tanto para los datos de entrenamiento como los de validación. Se comenzará probando con un tamaño de ventana 800 y paso de 250.

3. Generación de imágenes a partir de ventanas

Para este problema se probaron dos métodos de generación de imágenes.

Una de ellas consistió en la generación de espectrogramas a partir de cada ventana, lo cual se hizo con la función $stft$ de scipy. Esta función calcula la *Short-time Fourier transform* una matriz 2D de dimensiones 129 x 8, que corresponderían a la frecuencia x intervalos de tiempo. La transformada se calcula para cada canal de una ventana por separado, ya que si se intenta sumar las señales y calcular el espectrograma a partir de eso no se obtienen buenos resultados. Finalmente los 8 espectrogramas correspondientes a cada canal se agrupan en la última dimensión de una matriz de 4 dimensiones, resultando una matriz de: cantidad de ventanas x 129 (frecuencia) x 8 (tiempo) x 8 (cantidad de canales).

La otra forma fue sin aplicar ninguna transformación a las señales, sin embargo para esto se tuvieron que expandir las dimensiones, pues una red convolucional 2D necesita que cada dato ingresado posea 3 dimensiones y una ventana sin transformar posee tan solo dos dimensiones: la dimensión de tiempo (800) y la de canales (8). Así a cada ventana se le agregó una dimensión de tamaño 1, resultando finalmente en una matriz de tamaño: cantidad de ventanas x 1 x 800 (tiempo) x 8 (cantidad de canales), con lo que es una matriz apta para procesar en una red convolucional 2D.

Probando en Kaggle la opción que generó los mejores resultados fue la segunda, por lo que fue la escogida como la más apta para resolver el problema.

4. Estructura de red utilizada

En cuanto a la estructura de la red utilizada, se basó en trabajo realizado en el siguiente paper *"Hand Gesture Recognition Based on EMG Data: A Convolutional Neural Network Approach"*. Si bien se intentó implementar la misma red, debido a las diferencias en el tamaño de las matrices utilizadas en este trabajo y en el paper, no se pudo utilizar la misma estructura. De esta forma lo que se hizo, fue mantener una estructura semejante pero variando la cantidad de capas y el tamaño de los kernels.

Por otro lado, se probaron dos estructuras, una donde la última capa consistía en una convolución 2D con función de activación $softmax$, y otra donde la última capa era una Dense con función de activación $softmax$. La red que obtuvo mejores resultados fue la primera, por lo que se mantuvo dicha estructura. Así, la estructura de la red utilizada es la siguiente:

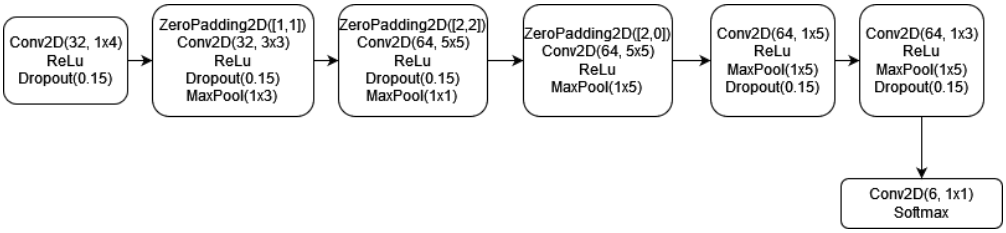

Figura 10: Estructura de red utilizada

Se utilizó el optimizador Adam y la función de pérdida *Categorical crossentropy*, no se probó variar estos parámetros porque siempre produjeron buenos resultados.

En cuanto a hiperparámetros, se varió el tamaño del batch entre 10, 64 y 100, y la cantidad de épocas entre 15, 25 y 30 dependiendo de las curvas de loss y accuracy que generan cada red.

La implementación de la red se hizo en la librería Keras.

5. Análisis preliminar eliminación y filtrado de imágenes

Dado que no se encontró un criterio que asegurara que realmente se estaba eliminando la mejor señal posible, se optó por probar en el set de validación el filtrado de cada una de las señales para discernir con las métricas de evaluación las mejores posibilidades.

Para definir qué señales eliminar, se utilizó la red que obtuvo los mejores resultados en Kaggle para probar qué resultados se obtenían luego de eliminar una señal. Esto también quiere decir que se utilizaron las señales sin espectrogramas. Así, se fueron eliminando las señales una por una para analizar las métricas de evaluación obtenidas, para finalmente pasar a probar el mejor métodos en términos de precisión de validación en Kaggle.

Según los resultados obtenidos en el set de validación, la eliminación de las señales 4 y 7 no lograban superar el 70 % de precisión, por lo que no se intentó subirlos a Kaggle. Del resto de las opciones, la que se veía más prometedora era la eliminación de la señal 6, que generó un accuracy en el set de validación cercano al 76 %, sin embargo al momento de subirlo a Kaggle solo logró un 50 %.

En Kaggle se probaron los modelos entrenados con datasets donde se filtraron los canales que generaron los mejores accuracies en validación, estos son el canal 1, 3, 5, 6, y 8. Cabe destacar que esta eliminación se hizo por separado, eliminando un canal a la vez.

Los resultados fueron los siguientes:

Señal filtrada	Public Score
Sin canal 1	73%
Sin canal 3	71%
Sin canal 5	66%
Sin canal 6	50%
Sin canal 8	38%

Figura 11: Public score de Kaggle sobre los modelos entrenados con eliminación de señales

A continuación se mostrarán los gráficos para la mejor de las opciones anteriores, que sería la eliminación del canal 1.

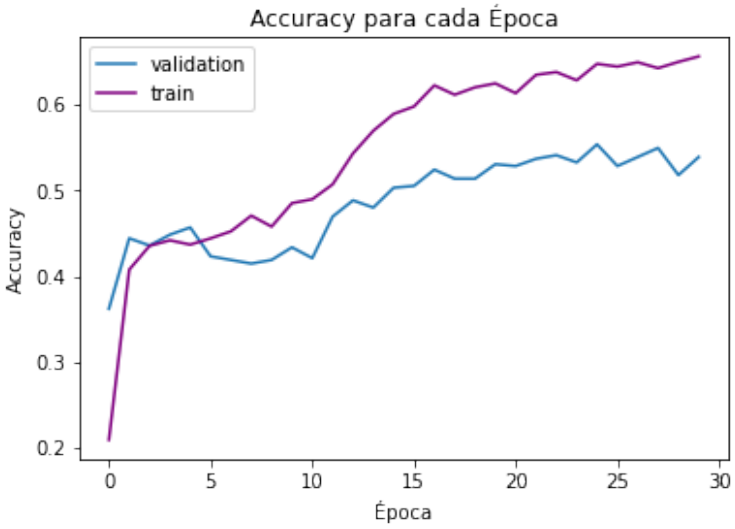

Figura 12: Precisión del conjunto de entrenamiento y validación durante entrenamiento para modelo sin señal 1

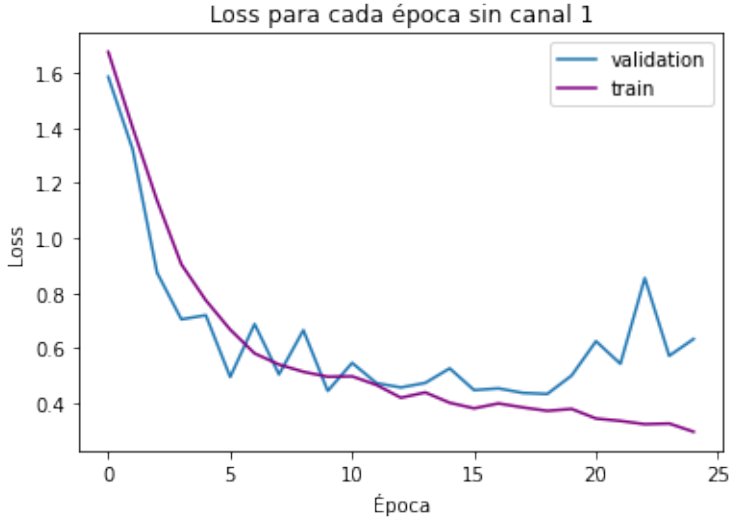

Figura 13: Pérdida del conjunto de entrenamiento y validación durante entrenamiento para modelo sin señal 1

Se puede ver que ambas curvas se ajustan, indicador de que no hay overfitting en este entrenamiento, así, si el modelo no genera mejores resultados no es debido a una mala elección de las épocas, sino que estarían afectando otras variables como por ejemplo, la estructura de la red.

En general, como se verá en la siguiente sección, no se lograron mejores puntajes que las redes sin eliminación de imágenes.

6. Resultados

Para presentar los resultados se mostrarán las matrices de confusión, gráfico de precisión y de pérdida para 4 variaciones de redes, con la capa dense, con la capa de convolución 2d, y con espectrogramas y sin él.

Se utilizaron nombres para cada una de las combinaciones. La red que ocupa la última capa convolución 2D y espectrogramas es la red 1, la semejante a la anterior pero que usa las ventanas de señales sin transformar es la red 2, la red que tiene una Dense de última capa y que recibe espectrogramas es la red 3 y la semejante que recibe las ventanas de señales es la red 4.

Se utiliza un 20 % de conjunto de validación, tamaño de batch 64 y la cantidad de épocas va variando, pues se intentó buscar para cada red la cantidad que generaba menos overfitting.

6.1. Conv2D con espectrogramas

Los hiperparámetros que se utilizan son: 64 para el tamaño del batch y 30 épocas.

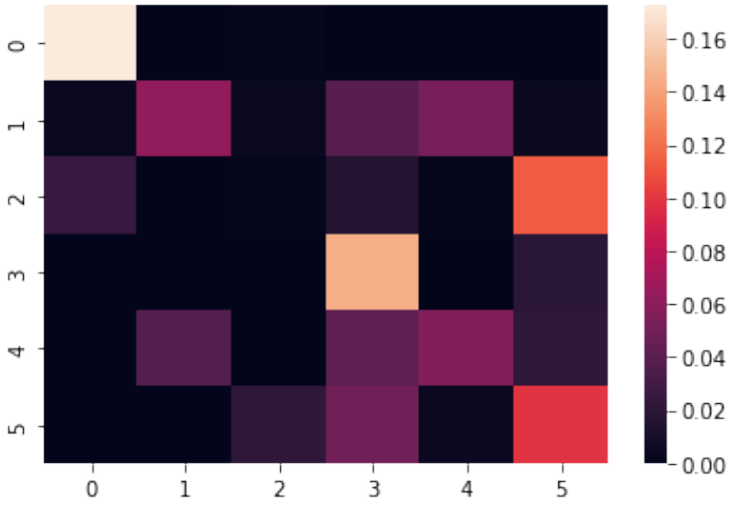

Figura 14: Matriz de confusión para la red sobre el conjunto de validación

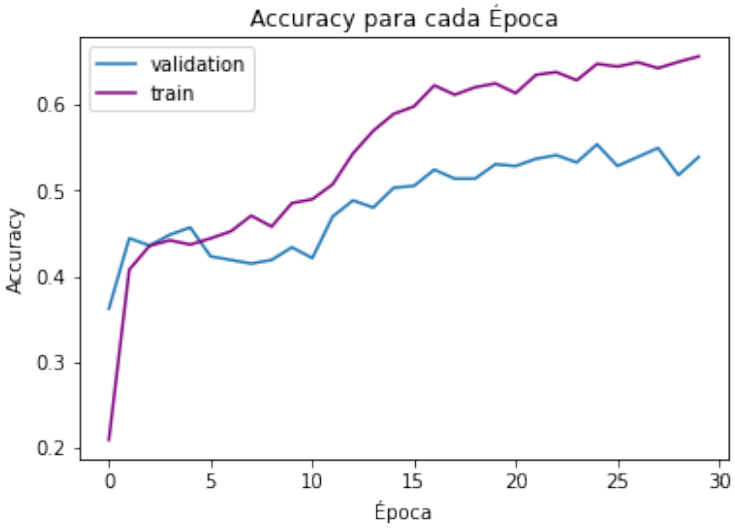

Figura 15: Precisión de la red 1 durante el entrenamiento

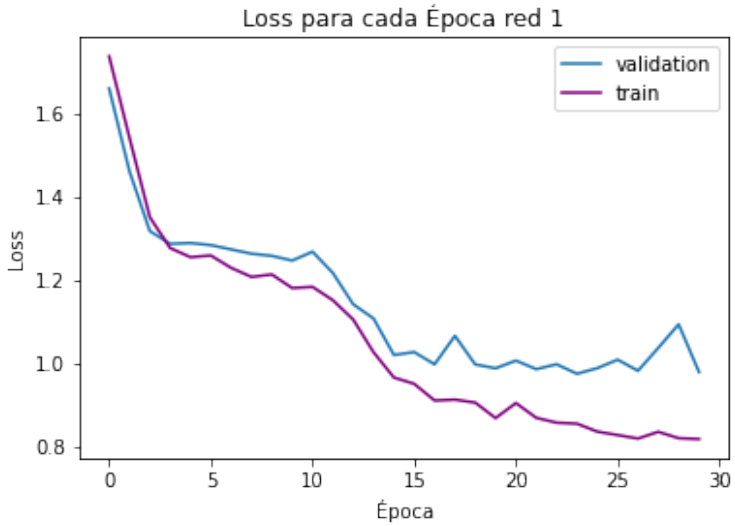

Figura 16: Pérdida de la red 1 durante el entrenamiento

6.2. Conv2D con señales

Los hiperparámetros que se utilizan son: 64 para el tamaño del batch y 25 épocas.

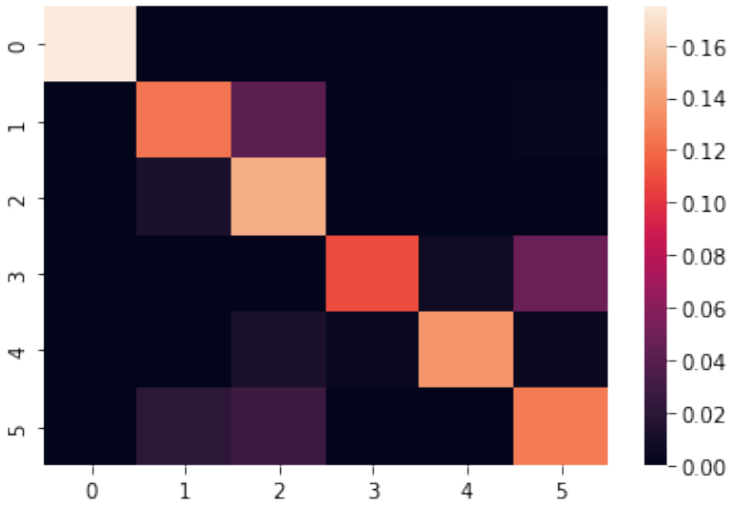

Figura 17: Matriz de confusión para la red sobre el conjunto de validación

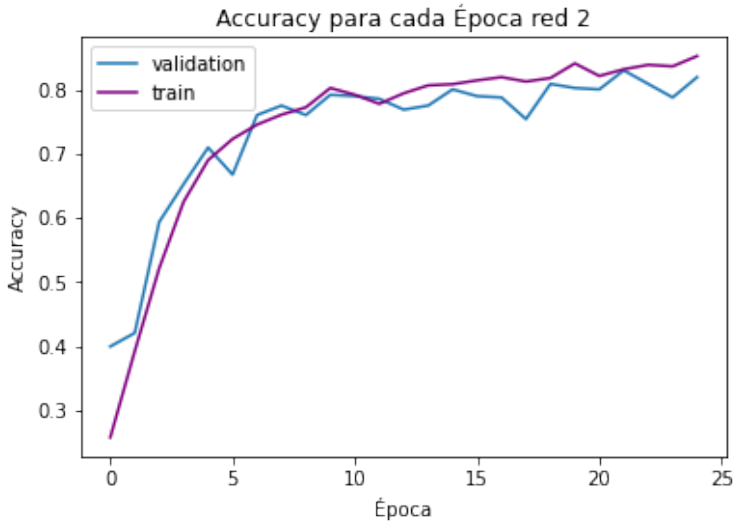

Figura 18: Precisión de la red 2 durante el entrenamiento

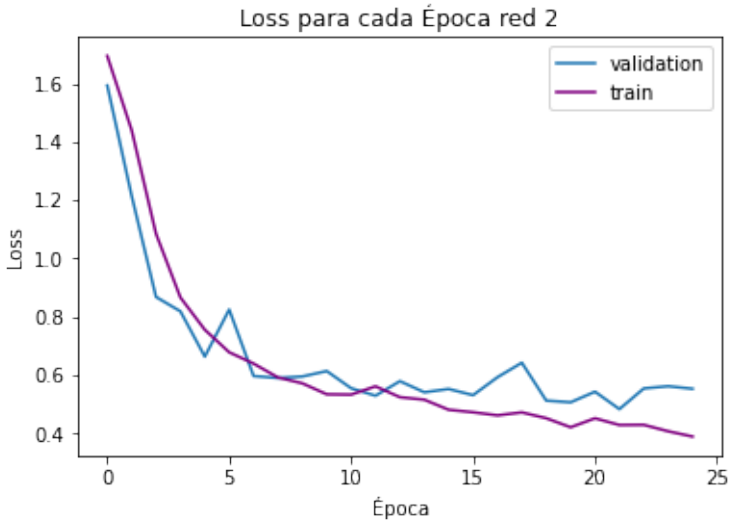

Figura 19: Pérdida de la red 2 durante el entrenamiento

6.3. Dense con espectrogramas

Los hiperparámetros que se utilizan son: 64 para el tamaño del batch y 35 épocas.

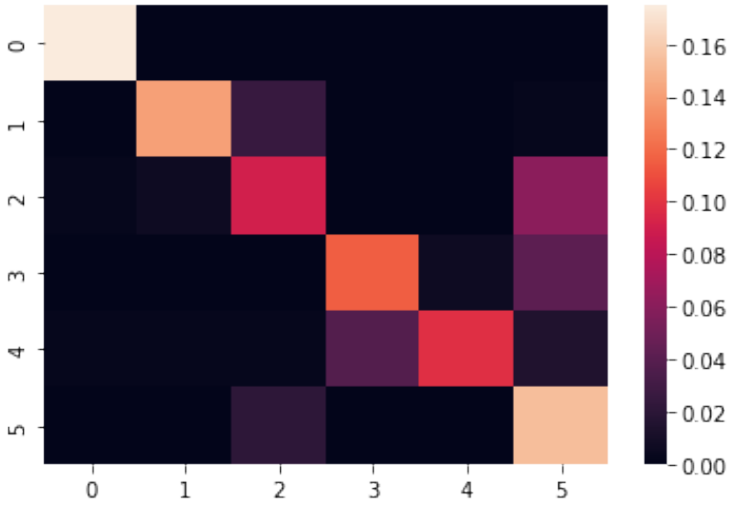

Figura 20: Matriz de confusión para la red sobre el conjunto de validación

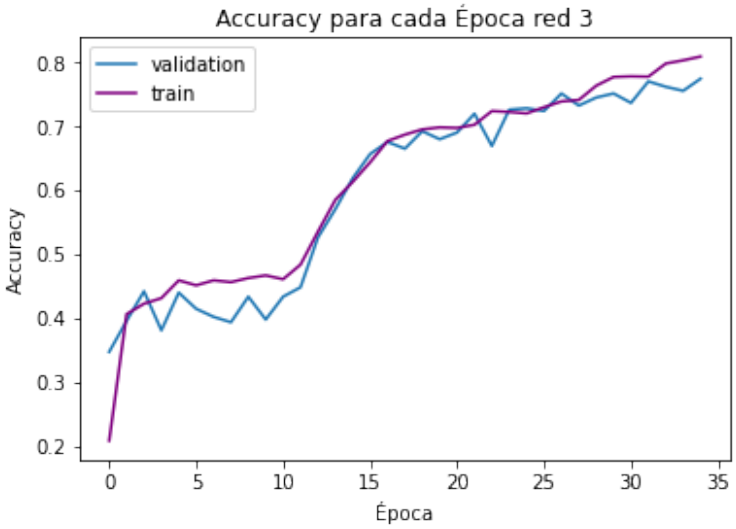

Figura 21: Precisión de la red 3 durante el entrenamiento

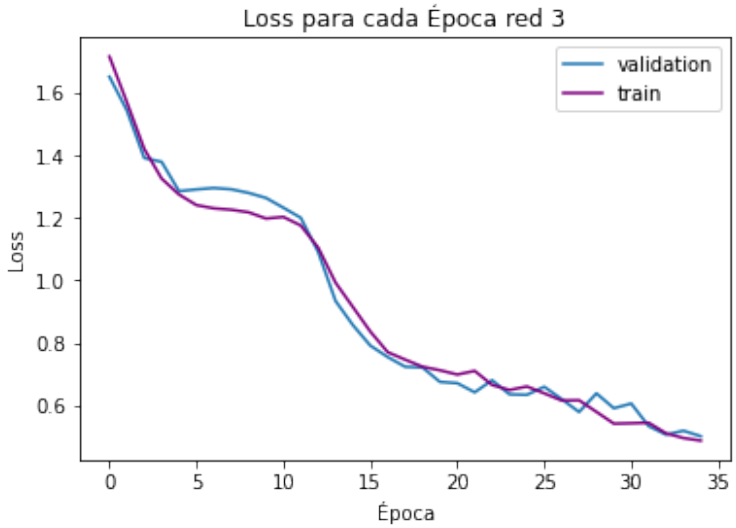

Figura 22: Pérdida de la red 3 durante el entrenamiento

6.4. Dense con señales

Los hiperparámetros que se utilizan son: 64 para el tamaño del batch y 30 épocas.

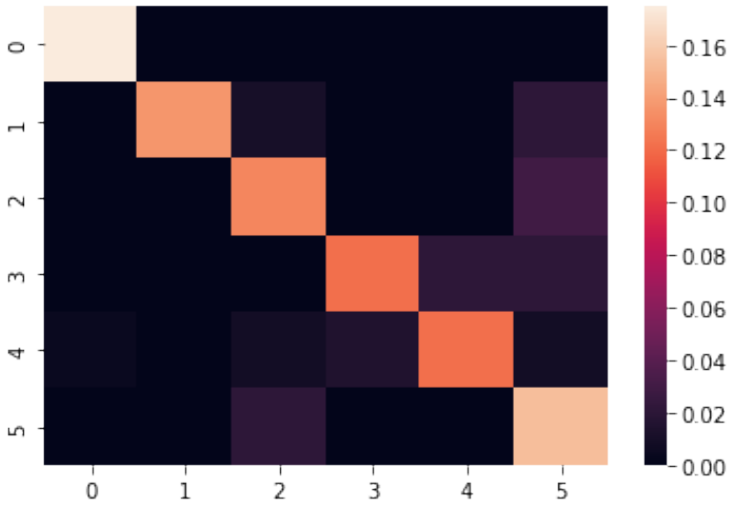

Figura 23: Matriz de confusión para la red sobre el conjunto de validación

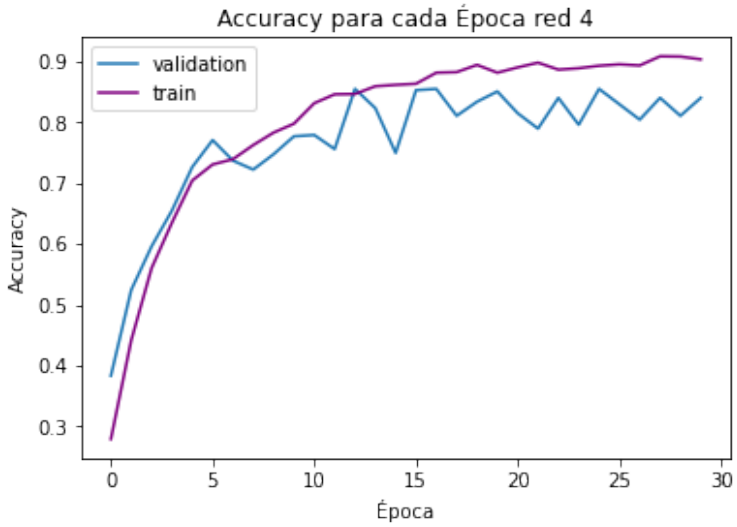

Figura 24: Precisión de la red 4 durante el entrenamiento

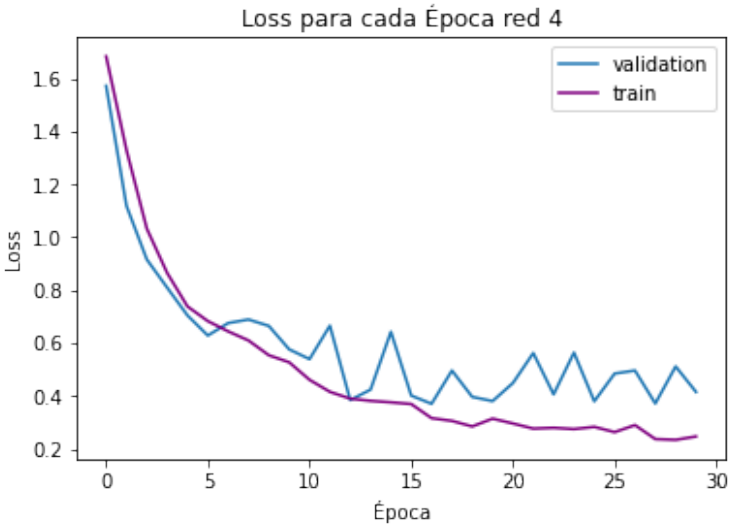

Figura 25: Pérdida de la red 4 durante el entrenamiento

De los resultados anteriores se puede notar que las opciones que trabajan con espectrogramas (red 1 y 3) son las que generan las peores matrices de confusión, indicando que cometen más errores al momento de predecir. Este resultado se condice con que las redes que hayan alcanzado una mejor precisión sean justamente las redes 2 y 4, que son las que ocupan las señales de las ventanas sin transformar.

Por otra parte, se puede notar que la matriz de confusión de la red que utiliza convolucion 2D en la última capa y las ventanas de señales (red 2) es mejor que su contraparte con última capa Dense (red 4). Aún así, ambas matrices son mejores que las que se generan con las redes que utilizan espectrogramas.

Finalmente, casi no hay overfitting en las opciones presentadas excepto por la primera, esto se puede ver en las curvas de precisión y de loss, donde las curvas de entrenamiento y validación se ajustan y no sucede que la de validación se aleje de la otra por una gran cantidad de épocas. Esto indica que la cantidad de épocas escogidas para cada red fue correcta.

7. Resultados en Kaggle

A continuación se presentan en una tabla los resultados obtenidos en Kaggle a partir de la variación de las redes, del pre-procesamiento de los datos, de los hiperparámetros y de las señales eliminadas. Los resultados están ordenados de mayor a menor public score.

Tipo de preprocesamiento	Tipo de última capa	Filtrado de señales	Porcentaje de validación	Épocas	Tamaño del batch	Public Score
Ventanas	Conv2D	NO	20%	25	64	0.81%
Ventanas	Conv2D	NO	20%	25	64	0.79%
Ventanas	Conv2D	NO	16%	30	64	0.789%
Ventanas	Conv2D	NO	20%	15	64	0.784%
Ventanas	Dense	NO	20%	30	64	0.75%
Ventanas	Conv2D	NO	23%	25	64	0.74%
Ventanas	Conv2D	NO	26%	30	64	0.73%
Ventanas	Conv2D	Señal 1	20%	25	64	73%
Espectrograma	Conv2D	NO	20%	30	64	0.72%
Ventanas	Conv2D	Señal 3	20%	25	64	71%
Espectrograma	Dense	NO	26%	35	64	0.70%
Espectrograma	Conv2D	NO	23%	25	64	0.68%
Espectrograma	Dense	NO	20%	30	64	0.67%
Ventanas	Conv2D	NO	20%	25	100	0.67%

Figura 26: Resultados de Kaggle ordenados por score obtenido

8. Análisis de resultados

Viendo los resultados en Kaggle se pueden sacar varias conclusiones. Disminuir el conjunto de validación a menos de 20 % empeora los resultados, esto se puede deber a que quedaban muy pocos datos para hacer el cross-validation y los resultados de las precisiones salían alterados. Es más, se pudo notar varias veces que en el conjunto de validación se generaban mejores resultados al momento de disminuir el porcentaje de validación, llegando incluso al 90 %, sin embargo al momento de probar en Kaggle el puntaje era mucho menor, apenas sobrepasando el 70 %.

Hacer que la última capa de la red sea una Dense empeora los resultados.

Los puntajes para los modelos que utilizan espectrogramas son peores que para los modelos que tan solo ocupan las señales sin pasar por una transformación. Esto es esperable dado los resultados que se obtenían durante el entrenmiento, que ya mostraban esta . Por esta misma razón, se hicieron pocas pruebas de este tipo en Kaggle.Es probable que se pierdan algunos datos por aplicar la transformada, o bien que la estructura de red escogida no se adapte bien al procesamiento de ellos.

El tamaño de batch genera mejores resultados cuando es 64, de hecho cuando se hizo la prueba con 100 el rendimiento disminuyó mucho.

La mejor cantidad de épocas varía entre cada modelo, por lo que es mejor calcularla a partir de los gráficos de accuracy y pérdidas generados durante el entrenamiento, esto fue lo que se hizo para

pobtener los mejores resultados de cada una de las redes.

No conviene hacer eliminación de imágenes, en este sentido, también es probable que utilizar esta técnica elimine información importante para la red al momento de clasificar. Ahora bien, dado que los mejores resultados se obtenían con las imágenes de las ventanas de señales, no se probó esta técnica con los espectrogramas, por lo que no se sabe si en realidad se podría adaptar mejor cuando se utilizan espectrogramas.

Dado que se hicieron muchas pruebas con muchas combinaciones de hiperparámetros distintas y no se logró mejorar el score por sobre el 81 % es posible que lo que en verdad se necesita cambiar para que esto suceda es la estructura de la red.

9. Problemas y posibles mejoras

Si bien se obtuvieron buenos resultados, se podrían aplicar mejoras al modelo. Una de ellas podría ser la evaluación de aplicar un filtro a las señales, por ejemplo aplicar un pasa-alta o un pasa-baja, lo cual en este trabajo no fue evaluado.

Además, se podría evaluar la red con eliminación de características sobre un espectrograma, ya que lo que pudo haber afectado es el hecho de trabajar con la señal. Sin embargo no es tan seguro que esta solución realmente aumente el score si es que este proceso elimina información importante para el clasificador.

Uno de los mayores problemas de utilizar redes convolucionales es que es muy complejo encontrar redes que se adapten adecuadamente al problema que se quiere resolver, e incluso si se encuentran trabajos asociados se corre el riesgo de que las dimensiones de los datos sean distintos y esto termina afectando a la estructura. Así, este fue el mayor problema presentado en el trabajo, y además se cree que fue la principal razón por la que no se logró un score más alto.

Finalmente, se podrían evaluar las redes utilizadas en otros trabajos o papers, cambiando más radicalmente la estructura generada. Esto también podría llevar a la creación de una red que trabaje mejor con espectrogramas. Se puede investigar qué estructuras de redes trabajan mejor datos del tipo imagen.

10. Conclusiones

Se puede concluir que con la red creada se puede resolver el problema satisfactoriamente, pues un accuracy de 81 % es bastante bueno. Sin embargo, viendo los resultados del resto del curso, se puede asegurar que se pueden encontrar redes que satisfagan mejor el problema. Así, se tendría que probar con algunas opciones que podrían mejorar los resultados obtenidos en este trabajo, de las cuales se cree que las principales son buscar una estructura de red que se adapte mejor a la clasificación de espectrogramas en vez de clasificar solo con las señales, estudiar el filtrado de señales y la reducción de ventanas.

En general, cabe notar que el problema del reconocimiento de gestos a partir de señales electromiográficas se puede resolver tanto con clasificadores como con redes convolucionales, esto se desprende de los resultados obtenidos por el curso.

Código .1: Insert code directly in your document

```
# −*− coding: utf−8 −*−
"""ProyectoIC(1)(1)(1)(2).ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/15−_yydpSDv7xWXCDFDDCrIat5ENGSHVa
"""

from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import random

train_df = pd.DataFrame()
val_df = pd.DataFrame()
subjects = []
df1 = pd.read_csv(
    '/content/drive/MyDrive/Dataset_de_senales_para_entrenamiento/emg_dataset/train/subj01/1.txt', sep="
      ↪ \t",header=None)
df2 = pd.read_csv(
    '/content/drive/MyDrive/Dataset_de_senales_para_entrenamiento/emg_dataset/train/subj01/2.txt', sep="
      ↪ \t",header=None)

val_people = 0
for i in range(1, 31):
  r = random.random()
  subject_route1 = ''
  subject_route2 = ''

  if i < 10:
    subject_route1 = '/content/drive/MyDrive/Dataset_de_senales_para_entrenamiento/emg_dataset/train/
      ↪ subj0'+ str(i) +'/1.txt'
    subject_route2 = '/content/drive/MyDrive/Dataset_de_senales_para_entrenamiento/emg_dataset/train/
      ↪ subj0'+ str(i) +'/2.txt'
  else :
    subject_route1 = '/content/drive/MyDrive/Dataset_de_senales_para_entrenamiento/emg_dataset/train/
      ↪ subj'+ str(i) +'/1.txt'
    subject_route2 = '/content/drive/MyDrive/Dataset_de_senales_para_entrenamiento/emg_dataset/train/
      ↪ subj'+ str(i) +'/2.txt'

  subject_df1 = pd.read_csv(subject_route1, sep="\t")
  subject_df2 = pd.read_csv(subject_route2, sep="\t")
  subjects.append(subject_df1)
  if r < 0.8:
    train_df = pd.concat([train_df, subject_df1, subject_df2], axis=0)
    train_df = train_df.reset_index(drop=True)
  else:
    val_people += 1
    val_df = pd.concat([val_df, subject_df1, subject_df2], axis=0)
    val_df = val_df.reset_index(drop=True)

print(len(train_df))
```

```
print(len(val_df))


print("Porcentaje de sujetos en el conjunto de validación:", val_people/30)
train_df

index0Names = train_df[ train_df['class'] == 0].index
index7Names = train_df[ train_df['class'] == 7].index
train_df.drop(index0Names , inplace=True)
train_df.drop(index7Names , inplace=True)
index0Names = val_df[ val_df['class'] == 0].index
index7Names = val_df[ val_df['class'] == 7].index
val_df.drop(index0Names , inplace=True)
val_df.drop(index7Names , inplace=True)
val_df = val_df.reset_index(drop=True)
train_df = train_df.reset_index(drop=True)
train_df

train_df.columns[1:8]

for column in train_df.columns[1:8]:
  train_df[column] = train_df[column] /train_df[column].abs().max()
  val_df[column] = val_df[column] /val_df[column].abs().max()

import numpy as np
import tensorflow as tf
import torch.nn as nn
from sklearn import preprocessing
from scipy.linalg import norm

def train_window_generator_matrix(df, w_size, w_step):
  #i ----> fila
  #j ----> columna
  id=0
  time = 0
  w_train_df = np.zeros([int(len(df)/800*4), 800, 8])
  r_train_df = []
  condition = True

  while condition:
    if time + 800 < len(df):
      first = df.loc[time]['class']
      last = df.loc[time+799]['class']

      if first  == last:
        #Si todos los elementos de la ventana son de la misma clase se hace la ventana
        #sino no se hace nada y se sigue con el ciclo.
        for channel in range(1, 9):
          window = df.loc[time:time+799]['channel'+str(channel)].to_numpy()

          #window_normed = window/np.absolute(window).max()

          for i in range(0,len(window)):
            w_train_df[id][i][channel-1] = window[i]
        id += 1
        class_ECO = np.zeros([1,6])
```

```
        class_ECO[0][int(first-1)] = 1
        r_train_df.append(class_ECO)

      time += w_step
    else:
      condition = False

  w_train_df = np.delete(w_train_df, range(id , len(w_train_df)), 0)
  return w_train_df, r_train_df

train_windows, train_classes = train_window_generator_matrix(train_df, 800, 250)
val_windows, val_classes = train_window_generator_matrix(val_df, 800, 250)

train_windows[0, 799, :]
train_classes[0]

print(train_windows.shape)
print(np.array(train_classes).shape)
print(val_windows.shape)
print(np.array(val_classes).shape)

#Poner los 8 canales en horizontal como columnas y los 800 del tamaño de la ventana hacia abajo (seria como el
    ↪ "time")
test_df = pd.read_csv(
  '/content/drive/MyDrive/Dataset_de_ventanas_de_prueba/emg_dataset/windows_test.csv', sep=",")

test_df

all = test_df.iloc[:,(1-1)*800+1:1*800 +1]
all_normed = all/all.abs().max()
test_df.iloc[:,(1-1)*800+1:1*800 +1] = all_normed

from typing_extensions import ParamSpecArgs
def test_dataset_generator_matrix(df):
  #normalizacion
  for channel in range(1, 9):
    all = df.iloc[:,(channel-1)*800+1 : channel*800 +1]
    all_normed = all/all.abs().max()
    df.iloc[:,(channel-1)*800+1 : channel*800 +1] = all_normed

  #i ----> fila
  #j ----> columna
  id=0
  #dataset sera una matriz de dimensiones: cantidad de ventanas x dimension ventana x cantidad canales
  w_test_df = np.zeros((len(df), 800, 8))
  condition = True
  while condition:
    if len(df) <= id:
      condition = False
    else:
      for channel in range(1,9):
        window = df.iloc[id,(channel-1)*800 + 1:channel*800 +1].to_numpy()
        for i in range(0,len(window)):
          w_test_df[id][i][channel-1] = window[i]
      id += 1

```

```
  return w_test_df
test_windows = test_dataset_generator_matrix(test_df)

"""#Gráfico de señales


"""

import matplotlib.pyplot as plt

x = subject_df1['time']
y = subject_df1['channel1']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 1 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel2']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 2 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel3']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 3 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel4']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 4 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel5']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 5 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel6']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
```

```
plt.title("Gráfico de señal de canal 6 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel7']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 7 para un sujeto")

x = subject_df1['time']
y = subject_df1['channel8']

plt.plot(x, y)
plt.xlabel("Tiempo")
plt.ylabel("Valores de señal")
plt.title("Gráfico de señal de canal 8 para un sujeto")

from scipy import signal
from scipy.fft import fftshift

x0 = train_windows[0, :, 0]
x1 = train_windows[0, :, 1]
x2 = train_windows[0, :, 2]
x3 = train_windows[0, :, 3]
x4 = train_windows[0, :, 4]
x5 = train_windows[0, :, 5]
x6 = train_windows[0, :, 6]
x7 = train_windows[0, :, 7]

x = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7
print(x.shape)
f, t, Zxx = signal.stft(x)

plt.pcolormesh(t, f, np.abs(Zxx), shading='gouraud')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()

from sklearn.decomposition import PCA
from sklearn import preprocessing
from scipy import signal

def specgram_generator2(df):
  input_df = np.zeros([len(df), 129, 8, 8], dtype='float')
  print(input_df.shape)
  for i in range(0, len(df)):
    x0 = df[i, :, 0]
    x1 = df[i, :, 1]
    x2 = df[i, :, 2]
    x3 = df[i, :, 3]
    x4 = df[i, :, 4]
    x5 = df[i, :, 5]
    x6 = df[i, :, 6]
    x7 = df[i, :, 7]

```

```
272
273     f0, t0, Sxx0 = signal.stft(x0)
274     f1, t1, Sxx1 = signal.stft(x1)
275     f2, t2, Sxx2 = signal.stft(x2)
276     f3, t3, Sxx3 = signal.stft(x3)
277     f4, t4, Sxx4 = signal.stft(x4)
278     f5, t5, Sxx5 = signal.stft(x5)
279     f6, t6, Sxx6 = signal.stft(x6)
280     f7, t7, Sxx7 = signal.stft(x7)
281
282     spec_list = [Sxx0, Sxx1, Sxx2, Sxx3, Sxx4, Sxx5, Sxx6, Sxx7]
283
284     for spec_i in range(0, len(spec_list)) :
285       #norms = np.linalg.norm(spec_list[spec_i])
286       input_df[i, : , : , spec_i] = spec_list[spec_i]
287
288   return input_df
289
290 train_input = specgram_generator2(train_windows)
291
292 val_input = specgram_generator2(val_windows)
293
294 val_windows.shape
295
296 """# Specgram
297
298 """
299
300 from tensorflow import keras
301 from keras import layers
302 from keras.models import Sequential
303 import torch
304 import torch.nn as nn
305 import torch.nn.functional as F
306 import torch.optim as optim
307 from torch.utils.data import Dataset, DataLoader
308 import numpy as np
309 from scipy.spatial import distance
310
311 import torchvision
312 import torchvision.transforms as transforms
313
314 #create model
315 model = Sequential()
316
317 model.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(129, 8 ,8),
        ↪ data_format="channels_last"))
318 model.add(layers.Dropout(0.15))
319
320 model.add(layers.ZeroPadding2D(padding=(1, 1)))
321 model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
322 model.add(layers.Dropout(0.15))
323 model.add(layers.MaxPooling2D(pool_size=(4, 3), padding='valid'))
324
325 model.add(layers.ZeroPadding2D(padding=(2, 2)))
326 model.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
```

```
model.add(layers.Dropout(0.15))
model.add(layers.MaxPooling2D(pool_size=(5, 1), padding='valid'))

model.add(layers.ZeroPadding2D(padding=(2, 0)))
model.add(layers.Conv2D(filters=64, kernel_size=(5, 1), activation="relu"))
model.add(layers.MaxPooling2D(pool_size=(5, 1), padding='valid'))
model.add(layers.Dropout(0.15))


model.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))



#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_input.shape)
print(np.array(train_classes).shape)
history = model.fit(train_input, np.array(train_classes).reshape(len(train_classes), 1, 1, 6), validation_data=(
    ↪ val_input, np.array(val_classes).reshape(len(val_classes), 1, 1, 6)),batch_size=batch_size, epochs=30)

plt.plot(range(0,len(history.history['val_accuracy'])), history.history['val_accuracy'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Accuracy")
plt.title("Accuracy para cada Época")
plt.plot(range(0,len(history.history['accuracy'])), history.history['accuracy'], label = "train", color = 'purple')
plt.legend()

plt.plot(range(0,len(history.history['val_loss'])), history.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada Época red 1")
plt.plot(range(0,len(history.history['loss'])), history.history['loss'], label = "train", color = 'purple')
plt.legend()

y = np.argmax(val_classes, axis=2)
y = y.reshape(len(val_classes),)
y

import seaborn as sns
from sklearn.metrics import confusion_matrix

y_pred = model.predict(val_input)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_input.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

"""#Test specgram"""

test_input = specgram_generator2(test_windows)
predictions = model.predict(test_input)
p = np.argmax(predictions, axis=3)
```

```
p = p.reshape(672,)

r = pd.DataFrame(columns = ['Id', 'Category'])
for i in range(0,len(p)):
  r.loc[i,'Id']=i
  r.loc[i,'Category']=p[i]+1

r

r.to_csv(index=False)

compression_opts = dict(method='zip',archive_name='out.csv')

r.to_csv('out.zip', index=False, compression=compression_opts)

"""#Ventanas sin procesar


"""

from tensorflow import keras
from keras import layers
from keras.models import Sequential
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
from scipy.spatial import distance

import torchvision
import torchvision.transforms as transforms

#create model
model2 = Sequential()

model2.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,8),
    ↪ data_format = "channels_last"))
model2.add(layers.Dropout(0.15))

model2.add(layers.ZeroPadding2D(padding=(1, 1)))
model2.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model2.add(layers.Dropout(0.15))
model2.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))

model2.add(layers.ZeroPadding2D(padding=(2, 2)))
model2.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model2.add(layers.Dropout(0.15))
model2.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))

model2.add(layers.ZeroPadding2D(padding=(2, 0)))
model2.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model2.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model2.add(layers.Dropout(0.15))

```

```
model2.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
model2.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model2.add(layers.Dropout(0.15))


model2.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
model2.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model2.add(layers.Dropout(0.15))

model2.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
#model2.add(layers.Dense(6, activation="softmax"))
model2.summary()

#compile model using accuracy to measure model performance
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

train_ex = np.expand_dims(train_windows, axis=0)
train_windows_r = train_windows.reshape(len(train_windows), 1, 800,8)
val_windows_r = val_windows.reshape(len(val_windows), 1, 800,8)

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_windows.shape)
print(np.array(train_classes).shape)
history2 = model2.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
    ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
    ↪ batch_size, epochs=25)

plt.plot(range(0,len(history2.history['val_accuracy'])), history2.history['val_accuracy'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Accuracy")
plt.title("Accuracy para cada Época red 2")
plt.plot(range(0,len(history2.history['accuracy'])), history2.history['accuracy'], label = "train", color = 'purple')
plt.legend()

plt.plot(range(0,len(history2.history['val_loss'])), history2.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada Época red 2")
plt.plot(range(0,len(history2.history['loss'])), history2.history['loss'], label = "train", color = 'purple')
plt.legend()

y_pred = model2.predict(val_windows_r)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_windows_r.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

#test_input = specgram_generator2(test_windows)
test_input = test_windows.reshape(len(test_windows), 1, 800,8)
predictions = model2.predict(test_input)
p = np.argmax(predictions, axis=3)
print(p.shape)
p = p.reshape(p.shape[0],)

```

```
r = pd.DataFrame(columns = ['Id', 'Category'])
for i in range(0,len(p)):
  r.loc[i,'Id']=i
  r.loc[i,'Category']=p[i]+1

r

compression_opts = dict(method='zip',archive_name='window.csv')

r.to_csv('window.zip', index=False, compression=compression_opts)

"""# Dense con specgrams

"""

from tensorflow import keras
from keras import layers
from keras.models import Sequential
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
from scipy.spatial import distance

import torchvision
import torchvision.transforms as transforms

#create model
model3 = Sequential()

model3.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(129, 8 ,8),
    ↪ data_format="channels_last"))
model3.add(layers.Dropout(0.15))

model3.add(layers.ZeroPadding2D(padding=(1, 1)))
model3.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model3.add(layers.Dropout(0.15))
model3.add(layers.MaxPooling2D(pool_size=(4, 3), padding='valid'))

model3.add(layers.ZeroPadding2D(padding=(2, 2)))
model3.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model3.add(layers.Dropout(0.15))
model3.add(layers.MaxPooling2D(pool_size=(5, 1), padding='valid'))

model3.add(layers.ZeroPadding2D(padding=(2, 0)))
model3.add(layers.Conv2D(filters=64, kernel_size=(5, 1), activation="relu"))
model3.add(layers.MaxPooling2D(pool_size=(5, 1), padding='valid'))
model3.add(layers.Dropout(0.15))


model3.add(layers.Dense(6, activation='softmax'))
```

```
#compile model using accuracy to measure model performance
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_input.shape)
print(np.array(train_classes).shape)
history3 = model3.fit(train_input, np.array(train_classes).reshape(len(train_classes), 1, 1, 6), validation_data
    ↪ =(val_input, np.array(val_classes).reshape(len(val_classes), 1, 1, 6)),batch_size=batch_size, epochs
    ↪ =35)

plt.plot(range(0,len(history3.history['val_accuracy'])), history3.history['val_accuracy'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Accuracy")
plt.title("Accuracy para cada Época red 3")
plt.plot(range(0,len(history3.history['accuracy'])), history3.history['accuracy'], label = "train", color = 'purple')
plt.legend()

plt.plot(range(0,len(history3.history['val_loss'])), history3.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada Época red 3")
plt.plot(range(0,len(history3.history['loss'])), history3.history['loss'], label = "train", color = 'purple')
plt.legend()

y_pred = model3.predict(val_input)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_input.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

test_input = specgram_generator2(test_windows)
predictions = model3.predict(test_input)
p = np.argmax(predictions, axis=3)
p = p.reshape(672,)

r = pd.DataFrame(columns = ['Id', 'Category'])
for i in range(0,len(p)):
  r.loc[i,'Id']=i
  r.loc[i,'Category']=p[i]+1

r

compression_opts = dict(method='zip',archive_name='dense.csv')

r.to_csv('dense.zip', index=False, compression=compression_opts)

"""# Dense de ventanas"""

from tensorflow import keras
from keras import layers
from keras.models import Sequential
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
from scipy.spatial import distance

import torchvision
import torchvision.transforms as transforms

model4 = Sequential()

model4.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,8),
    ↪ data_format = "channels_last"))
model4.add(layers.Dropout(0.15))

model4.add(layers.ZeroPadding2D(padding=(1, 1)))
model4.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model4.add(layers.Dropout(0.15))
model4.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))

model4.add(layers.ZeroPadding2D(padding=(2, 2)))
model4.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model4.add(layers.Dropout(0.15))
model4.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))

model4.add(layers.ZeroPadding2D(padding=(2, 0)))
model4.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model4.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model4.add(layers.Dropout(0.15))

model4.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
model4.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model4.add(layers.Dropout(0.15))


model4.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
model4.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model4.add(layers.Dropout(0.15))



model4.add(layers.Dense(6, activation='softmax'))



#compile model using accuracy to measure model performance
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

train_ex = np.expand_dims(train_windows, axis=0)
train_windows_r = train_windows.reshape(len(train_windows), 1, 800,8)
val_windows_r = val_windows.reshape(len(val_windows), 1, 800,8)

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_windows.shape)
print(np.array(train_classes).shape)
```

```
655 history4 = model4.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
        ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
        ↪ batch_size, epochs=30)
656
657 plt.plot(range(0,len(history4.history['val_accuracy'])), history4.history['val_accuracy'], label = "validation")
658 plt.xlabel("Época")
659 plt.ylabel("Accuracy")
660 plt.title("Accuracy para cada Época red 4")
661 plt.plot(range(0,len(history4.history['accuracy'])), history4.history['accuracy'], label = "train", color = 'purple')
662 plt.legend()
663
664 plt.plot(range(0,len(history4.history['val_loss'])), history4.history['val_loss'], label = "validation")
665 plt.xlabel("Época")
666 plt.ylabel("Loss")
667 plt.title("Loss para cada Época red 4")
668 plt.plot(range(0,len(history4.history['loss'])), history4.history['loss'], label = "train", color = 'purple')
669 plt.legend()
670
671 y_pred = model4.predict(val_windows_r)
672 y_pred = np.argmax(y_pred, axis=3)
673 y_pred = y_pred.reshape(val_windows_r.shape[0],)
674
675 sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')
676
677 #test_input = specgram_generator2(test_windows)
678 test_input = test_windows.reshape(len(test_windows), 1, 800,8)
679 predictions = model4.predict(test_input)
680 p = np.argmax(predictions, axis=3)
681 print(p.shape)
682 p = p.reshape(p.shape[0],)
683
684 r = pd.DataFrame(columns = ['Id', 'Category'])
685 for i in range(0,len(p)):
686   r.loc[i,'Id']=i
687   r.loc[i,'Category']=p[i]+1
688
689 r
690
691 compression_opts = dict(method='zip',archive_name='dense-window.csv')
692
693 r.to_csv('dense-window.zip', index=False, compression=compression_opts)
694
695 """# Eliminación de señales"""
696
697 x = subject_df1['time']
698 y1 = subject_df1['channel1']
699 y2 = subject_df1['channel2']
700 y3 = subject_df1['channel3']
701 y4 = subject_df1['channel4']
702 y5 = subject_df1['channel5']
703 y6 = subject_df1['channel6']
704 y7 = subject_df1['channel7']
705 y8 = subject_df1['channel8']
706
707 plt.plot(x, y1, label = "channel1")
708 plt.plot(x, y2, label = "channel2", color = 'purple')
```

```
plt.plot(x, y3, label = "channel3", color = 'red')
plt.plot(x, y4, label = "channel4", color = 'orange')
plt.plot(x, y5, label = "channel5", color = 'yellow')
plt.plot(x, y6, label = "channel6", color = 'black')
plt.plot(x, y7, label = "channel7", color = 'gray')
plt.plot(x, y8, label = "channel8", color = 'green')

plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada Época red 1")
plt.legend()

x = subjects[0]['time']
y1 = subjects[0]['channel1']
y2 = subjects[0]['channel2']
y3 = subjects[0]['channel3']
y4 = subjects[0]['channel4']
y5 = subjects[0]['channel5']
y6 = subjects[0]['channel6']
y7 = subjects[0]['channel7']
y8 = subjects[0]['channel8']

plt.plot(x, y1, label = "channel1")
plt.plot(x, y2, label = "channel2", color = 'purple')
plt.plot(x, y3, label = "channel3", color = 'red')
plt.plot(x, y4, label = "channel4", color = 'orange')
plt.plot(x, y5, label = "channel5", color = 'yellow')
plt.plot(x, y6, label = "channel6", color = 'black')
plt.plot(x, y7, label = "channel7", color = 'gray')
plt.plot(x, y8, label = "channel8", color = 'green')

plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada Época red 1")
plt.legend()

"""## Menos chanel 1

"""

def train_window_generator_matrix_2(df, w_size, w_step, w_channel):
  #i ----> fila
  #j ----> columna
  id=0
  time = 0
  w_train_df = np.zeros([int(len(df)/800*4), 800, 7])
  r_train_df = []
  condition = True

  while condition:
    if time + 800 < len(df):
      first = df.loc[time]['class']
      last = df.loc[time+799]['class']

      if first  == last:
        #Si todos los elementos de la ventana son de la misma clase se hace la ventana
```

```
765         #sino no se hace nada y se sigue con el ciclo.
766         for channel in range(1, 9):
767           if channel != w_channel:
768             window = df.loc[time:time+799]['channel'+str(channel)].to_numpy()
769             for i in range(0,len(window)):
770               if channel < w_channel:
771                 w_train_df[id][i][channel-1] = window[i]
772               else:
773                 w_train_df[id][i][channel-2] = window[i]
774             w_train_df[id][i][channel-2] = window[i]
775         id += 1
776         class_ECO = np.zeros([1,6])
777         class_ECO[0][int(first-1)] = 1
778         r_train_df.append(class_ECO)
779 
780       time += w_step
781     else:
782       condition = False
783 
784   w_train_df = np.delete(w_train_df, range(id , len(w_train_df)), 0)
785   return w_train_df, r_train_df
786 
787 def specgram_generator3(df):
788   input_df = np.zeros([len(df), 129, 8, 8], dtype='float')
789   print(input_df.shape)
790   for i in range(0, len(df)):
791     x0 = df[i, :, 0]
792     x1 = df[i, :, 1]
793     x2 = df[i, :, 2]
794     x3 = df[i, :, 3]
795     x4 = df[i, :, 4]
796     x5 = df[i, :, 5]
797     x6 = df[i, :, 6]
798 
799 
800 
801     f0, t0, Sxx0 = signal.stft(x0)
802     f1, t1, Sxx1 = signal.stft(x1)
803     f2, t2, Sxx2 = signal.stft(x2)
804     f3, t3, Sxx3 = signal.stft(x3)
805     f4, t4, Sxx4 = signal.stft(x4)
806     f5, t5, Sxx5 = signal.stft(x5)
807     f6, t6, Sxx6 = signal.stft(x6)
808 
809     spec_list = [Sxx0, Sxx1, Sxx2, Sxx3, Sxx4, Sxx5, Sxx6]
810 
811     for spec_i in range(0, len(spec_list)) :
812       #norms = np.linalg.norm(spec_list[spec_i])
813       input_df[i, : , : , spec_i] = spec_list[spec_i]
814 
815   return input_df
816 
817 train_df_f = train_df.drop(columns=['channel1'])
818 val_df_f = val_df.drop(columns=['channel1'])
819 train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,1)
820 val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,1)
```

```
821
822 #create model
823 model6 = Sequential()
824
825 model6.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
        ↪ data_format = "channels_last"))
826 model6.add(layers.Dropout(0.15))
827
828 model6.add(layers.ZeroPadding2D(padding=(1, 1)))
829 model6.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
830 model6.add(layers.Dropout(0.15))
831 model6.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))
832
833 model6.add(layers.ZeroPadding2D(padding=(2, 2)))
834 model6.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
835 model6.add(layers.Dropout(0.15))
836 model6.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))
837
838 model6.add(layers.ZeroPadding2D(padding=(2, 0)))
839 model6.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
840 model6.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
841 model6.add(layers.Dropout(0.15))
842
843 model6.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
844 model6.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
845 model6.add(layers.Dropout(0.15))
846
847
848 model6.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
849 model6.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
850 model6.add(layers.Dropout(0.15))
851
852
853 model6.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
854 #model2.add(layers.Dense(6, activation="softmax"))
855 model6.summary()
856
857 #compile model using accuracy to measure model performance
858 model6.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
859
860 train_ex = np.expand_dims(train_windows, axis=0)
861 train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
862 val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)
863
864 # Train the model for 1 epoch from Numpy data
865 batch_size = 64
866 print("Fit on NumPy data")
867 print(train_windows.shape)
868 print(np.array(train_classes).shape)
869 history6 = model6.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
        ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
        ↪ batch_size, epochs=25)
870
871 y_pred = model6.predict(val_windows_r)
872 y_pred = np.argmax(y_pred, axis=3)
873 y_pred = y_pred.reshape(val_windows_r.shape[0],)
```

```
874
875 sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')
876
877 plt.plot(range(0,len(history6.history['val_accuracy'])), history6.history['val_accuracy'], label = "validation")
878 plt.xlabel("Época")
879 plt.ylabel("Accuracy")
880 plt.title("Accuracy para cada época sin canal 1")
881 plt.plot(range(0,len(history6.history['accuracy'])), history6.history['accuracy'], label = "train", color = 'purple')
882 plt.legend()
883
884 plt.plot(range(0,len(history6.history['val_loss'])), history6.history['val_loss'], label = "validation")
885 plt.xlabel("Época")
886 plt.ylabel("Loss")
887 plt.title("Loss para cada época sin canal 1")
888 plt.plot(range(0,len(history6.history['loss'])), history6.history['loss'], label = "train", color = 'purple')
889 plt.legend()
890
891 """## Sin canal 2"""
892
893 train_df_f = train_df.drop(columns=['channel2'])
894 val_df_f = val_df.drop(columns=['channel2'])
895 train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,2)
896 val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,2)
897
898 #create model
899 model7 = Sequential()
900
901 model7.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
        ↪ data_format = "channels_last"))
902 model7.add(layers.Dropout(0.15))
903
904 model7.add(layers.ZeroPadding2D(padding=(1, 1)))
905 model7.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
906 model7.add(layers.Dropout(0.15))
907 model7.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))
908
909 model7.add(layers.ZeroPadding2D(padding=(2, 2)))
910 model7.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
911 model7.add(layers.Dropout(0.15))
912 model7.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))
913
914 model7.add(layers.ZeroPadding2D(padding=(2, 0)))
915 model7.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
916 model7.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
917 model7.add(layers.Dropout(0.15))
918
919 model7.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
920 model7.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
921 model7.add(layers.Dropout(0.15))
922
923
924 model7.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
925 model7.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
926 model7.add(layers.Dropout(0.15))
927
928
```

```
929 model7.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
930 #model2.add(layers.Dense(6, activation="softmax"))
931 model7.summary()
932
933 #compile model using accuracy to measure model performance
934 model7.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
935
936 train_ex = np.expand_dims(train_windows, axis=0)
937 train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
938 val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)
939
940 # Train the model for 1 epoch from Numpy data
941 batch_size = 64
942 print("Fit on NumPy data")
943 print(train_windows.shape)
944 print(np.array(train_classes).shape)
945 history7 = model7.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
        ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
        ↪ batch_size, epochs=25)
946
947 y_pred = model7.predict(val_windows_r)
948 y_pred = np.argmax(y_pred, axis=3)
949 y_pred = y_pred.reshape(val_windows_r.shape[0],)
950
951 sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')
952
953 plt.plot(range(0,len(history7.history['val_accuracy'])), history7.history['val_accuracy'], label = "validation")
954 plt.xlabel("Época")
955 plt.ylabel("Accuracy")
956 plt.title("Accuracy para cada época sin canal 1")
957 plt.plot(range(0,len(history7.history['accuracy'])), history7.history['accuracy'], label = "train", color = 'purple')
958 plt.legend()
959
960 plt.plot(range(0,len(history7.history['val_loss'])), history7.history['val_loss'], label = "validation")
961 plt.xlabel("Época")
962 plt.ylabel("Loss")
963 plt.title("Loss para cada época sin canal 1")
964 plt.plot(range(0,len(history7.history['loss'])), history7.history['loss'], label = "train", color = 'purple')
965 plt.legend()
966
967 """## Sin canal 3"""
968
969 train_df_f = train_df.drop(columns=['channel3'])
970 val_df_f = val_df.drop(columns=['channel3'])
971 train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,3)
972 val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,3)
973
974 #create model
975 model8 = Sequential()
976
977 model8.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
        ↪ data_format = "channels_last"))
978 model8.add(layers.Dropout(0.15))
979
980 model8.add(layers.ZeroPadding2D(padding=(1, 1)))
981 model8.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
```

```
982 model8.add(layers.Dropout(0.15))
983 model8.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))
984
985 model8.add(layers.ZeroPadding2D(padding=(2, 2)))
986 model8.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
987 model8.add(layers.Dropout(0.15))
988 model8.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))
989
990 model8.add(layers.ZeroPadding2D(padding=(2, 0)))
991 model8.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
992 model8.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
993 model8.add(layers.Dropout(0.15))
994
995 model8.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
996 model8.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
997 model8.add(layers.Dropout(0.15))
998
999
1000 model8.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
1001 model8.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
1002 model8.add(layers.Dropout(0.15))
1003
1004
1005 model8.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
1006 #model2.add(layers.Dense(6, activation="softmax"))
1007 model8.summary()
1008
1009 #compile model using accuracy to measure model performance
1010 model8.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
1011
1012 train_ex = np.expand_dims(train_windows, axis=0)
1013 train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
1014 val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)
1015
1016 # Train the model for 1 epoch from Numpy data
1017 batch_size = 64
1018 print("Fit on NumPy data")
1019 print(train_windows.shape)
1020 print(np.array(train_classes).shape)
1021 history8 = model8.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
      ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
      ↪ batch_size, epochs=25)
1022
1023 y_pred = model8.predict(val_windows_r)
1024 y_pred = np.argmax(y_pred, axis=3)
1025 y_pred = y_pred.reshape(val_windows_r.shape[0],)
1026
1027 sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')
1028
1029 plt.plot(range(0,len(history8.history['val_accuracy'])), history8.history['val_accuracy'], label = "validation")
1030 plt.xlabel("Época")
1031 plt.ylabel("Accuracy")
1032 plt.title("Accuracy para cada época sin canal 1")
1033 plt.plot(range(0,len(history8.history['accuracy'])), history8.history['accuracy'], label = "train", color = 'purple')
1034 plt.legend()
1035
```

```
plt.plot(range(0,len(history8.history['val_loss'])), history8.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada época sin canal 1")
plt.plot(range(0,len(history8.history['loss'])), history8.history['loss'], label = "train", color = 'purple')
plt.legend()

"""## Sin canal 4"""

train_df_f = train_df.drop(columns=['channel4'])
val_df_f = val_df.drop(columns=['channel4'])
train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,4)
val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,4)

#create model
model9 = Sequential()

model9.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
    ↪ data_format = "channels_last"))
model9.add(layers.Dropout(0.15))

model9.add(layers.ZeroPadding2D(padding=(1, 1)))
model9.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model9.add(layers.Dropout(0.15))
model9.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))

model9.add(layers.ZeroPadding2D(padding=(2, 2)))
model9.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model9.add(layers.Dropout(0.15))
model9.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))

model9.add(layers.ZeroPadding2D(padding=(2, 0)))
model9.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model9.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model9.add(layers.Dropout(0.15))

model9.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
model9.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model9.add(layers.Dropout(0.15))


model9.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
model9.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model9.add(layers.Dropout(0.15))


model9.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
#model2.add(layers.Dense(6, activation="softmax"))
model9.summary()

#compile model using accuracy to measure model performance
model9.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

train_ex = np.expand_dims(train_windows, axis=0)
train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)
```

```
1091
1092 # Train the model for 1 epoch from Numpy data
1093 batch_size = 64
1094 print("Fit on NumPy data")
1095 print(train_windows.shape)
1096 print(np.array(train_classes).shape)
1097 history9 = model9.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
      ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
      ↪ batch_size, epochs=25)
1098
1099 y_pred = model9.predict(val_windows_r)
1100 y_pred = np.argmax(y_pred, axis=3)
1101 y_pred = y_pred.reshape(val_windows_r.shape[0],)
1102
1103 sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')
1104
1105 plt.plot(range(0,len(history9.history['val_accuracy'])), history9.history['val_accuracy'], label = "validation")
1106 plt.xlabel("Época")
1107 plt.ylabel("Accuracy")
1108 plt.title("Accuracy para cada época sin canal 1")
1109 plt.plot(range(0,len(history9.history['accuracy'])), history9.history['accuracy'], label = "train", color = 'purple')
1110 plt.legend()
1111
1112 plt.plot(range(0,len(history9.history['val_loss'])), history9.history['val_loss'], label = "validation")
1113 plt.xlabel("Época")
1114 plt.ylabel("Loss")
1115 plt.title("Loss para cada época sin canal 1")
1116 plt.plot(range(0,len(history9.history['loss'])), history9.history['loss'], label = "train", color = 'purple')
1117 plt.legend()
1118
1119 """## Sin canal 5"""
1120
1121 train_df_f = train_df.drop(columns=['channel5'])
1122 val_df_f = val_df.drop(columns=['channel5'])
1123 train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,5)
1124 val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,5)
1125
1126 #create model
1127 model10 = Sequential()
1128
1129 model10.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
      ↪ data_format = "channels_last"))
1130 model10.add(layers.Dropout(0.15))
1131
1132 model10.add(layers.ZeroPadding2D(padding=(1, 1)))
1133 model10.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
1134 model10.add(layers.Dropout(0.15))
1135 model10.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))
1136
1137 model10.add(layers.ZeroPadding2D(padding=(2, 2)))
1138 model10.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
1139 model10.add(layers.Dropout(0.15))
1140 model10.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))
1141
1142 model10.add(layers.ZeroPadding2D(padding=(2, 0)))
1143 model10.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
```

```
model10.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model10.add(layers.Dropout(0.15))

model10.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
model10.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model10.add(layers.Dropout(0.15))


model10.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
model10.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model10.add(layers.Dropout(0.15))


model10.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
#model2.add(layers.Dense(6, activation="softmax"))
model10.summary()

#compile model using accuracy to measure model performance
model10.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

train_ex = np.expand_dims(train_windows, axis=0)
train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_windows.shape)
print(np.array(train_classes).shape)
history10 = model10.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
    ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
    ↪ batch_size, epochs=25)

y_pred = model10.predict(val_windows_r)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_windows_r.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

plt.plot(range(0,len(history10.history['val_accuracy'])), history10.history['val_accuracy'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Accuracy")
plt.title("Accuracy para cada época sin canal 1")
plt.plot(range(0,len(history10.history['accuracy'])), history10.history['accuracy'], label = "train", color = 'purple')
plt.legend()

plt.plot(range(0,len(history10.history['val_loss'])), history10.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada época sin canal 1")
plt.plot(range(0,len(history10.history['loss'])), history10.history['loss'], label = "train", color = 'purple')
plt.legend()

"""## Sin canal 6 """

train_df_f = train_df.drop(columns=['channel6'])
```

```
val_df_f = val_df.drop(columns=['channel6'])
train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,6)
val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,6)

#create model
model11 = Sequential()

model11.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
    ↪ data_format = "channels_last"))
model11.add(layers.Dropout(0.15))

model11.add(layers.ZeroPadding2D(padding=(1, 1)))
model11.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model11.add(layers.Dropout(0.15))
model11.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))

model11.add(layers.ZeroPadding2D(padding=(2, 2)))
model11.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model11.add(layers.Dropout(0.15))
model11.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))

model11.add(layers.ZeroPadding2D(padding=(2, 0)))
model11.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model11.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model11.add(layers.Dropout(0.15))

model11.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
model11.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model11.add(layers.Dropout(0.15))


model11.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
model11.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model11.add(layers.Dropout(0.15))


model11.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
#model2.add(layers.Dense(6, activation="softmax"))
model11.summary()

#compile model using accuracy to measure model performance
model11.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

train_ex = np.expand_dims(train_windows, axis=0)
train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_windows.shape)
print(np.array(train_classes).shape)
history11 = model11.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
    ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
    ↪ batch_size, epochs=25)

```

```
y_pred = model11.predict(val_windows_r)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_windows_r.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

plt.plot(range(0,len(history11.history['val_accuracy'])), history11.history['val_accuracy'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Accuracy")
plt.title("Accuracy para cada época sin canal 6")
plt.plot(range(0,len(history11.history['accuracy'])), history11.history['accuracy'], label = "train", color = 'purple')
plt.legend()

plt.plot(range(0,len(history11.history['val_loss'])), history11.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada época sin canal 6")
plt.plot(range(0,len(history11.history['loss'])), history11.history['loss'], label = "train", color = 'purple')
plt.legend()

"""## Sin canal 7"""

train_df_f = train_df.drop(columns=['channel7'])
val_df_f = val_df.drop(columns=['channel7'])
train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,7)
val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,7)

#create model
model12 = Sequential()

model12.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
    ↪ data_format = "channels_last"))
model12.add(layers.Dropout(0.15))

model12.add(layers.ZeroPadding2D(padding=(1, 1)))
model12.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model12.add(layers.Dropout(0.15))
model12.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))

model12.add(layers.ZeroPadding2D(padding=(2, 2)))
model12.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model12.add(layers.Dropout(0.15))
model12.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))

model12.add(layers.ZeroPadding2D(padding=(2, 0)))
model12.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model12.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model12.add(layers.Dropout(0.15))

model12.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
model12.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
model12.add(layers.Dropout(0.15))


model12.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
model12.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
```

```
model12.add(layers.Dropout(0.15))


model12.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
#model2.add(layers.Dense(6, activation="softmax"))
model12.summary()

#compile model using accuracy to measure model performance
model12.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

train_ex = np.expand_dims(train_windows, axis=0)
train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)

# Train the model for 1 epoch from Numpy data
batch_size = 64
print("Fit on NumPy data")
print(train_windows.shape)
print(np.array(train_classes).shape)
history12 = model12.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
    ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
    ↪ batch_size, epochs=25)

y_pred = model12.predict(val_windows_r)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_windows_r.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

plt.plot(range(0,len(history12.history['val_accuracy'])), history12.history['val_accuracy'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Accuracy")
plt.title("Accuracy para cada época sin canal 1")
plt.plot(range(0,len(history12.history['accuracy'])), history12.history['accuracy'], label = "train", color = 'purple')
plt.legend()

plt.plot(range(0,len(history12.history['val_loss'])), history12.history['val_loss'], label = "validation")
plt.xlabel("Época")
plt.ylabel("Loss")
plt.title("Loss para cada época sin canal 1")
plt.plot(range(0,len(history12.history['loss'])), history12.history['loss'], label = "train", color = 'purple')
plt.legend()

"""## Sin canal 8"""

train_df_f = train_df.drop(columns=['channel8'])
val_df_f = val_df.drop(columns=['channel8'])
train_windows, train_classes = train_window_generator_matrix_2(train_df_f, 800, 250,8)
val_windows, val_classes = train_window_generator_matrix_2(val_df_f, 800, 250,8)

#create model
model13 = Sequential()

model13.add(layers.Conv2D(filters=32, kernel_size=(1, 4), activation="relu", input_shape=(1,800 ,7),
    ↪ data_format = "channels_last"))
model13.add(layers.Dropout(0.15))
```

```
1359
1360 model13.add(layers.ZeroPadding2D(padding=(1, 1)))
1361 model13.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
1362 model13.add(layers.Dropout(0.15))
1363 model13.add(layers.MaxPooling2D(pool_size=(1, 3), padding='valid'))
1364
1365 model13.add(layers.ZeroPadding2D(padding=(2, 2)))
1366 model13.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
1367 model13.add(layers.Dropout(0.15))
1368 model13.add(layers.MaxPooling2D(pool_size=(1, 1), padding='valid'))
1369
1370 model13.add(layers.ZeroPadding2D(padding=(2, 0)))
1371 model13.add(layers.Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
1372 model13.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
1373 model13.add(layers.Dropout(0.15))
1374
1375 model13.add(layers.Conv2D(filters=64, kernel_size=(1, 5), activation="relu"))
1376 model13.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
1377 model13.add(layers.Dropout(0.15))
1378
1379
1380 model13.add(layers.Conv2D(filters=64, kernel_size=(1, 3), activation="relu"))
1381 model13.add(layers.MaxPooling2D(pool_size=(1, 5), padding='valid'))
1382 model13.add(layers.Dropout(0.15))
1383
1384
1385 model13.add(layers.Conv2D(filters=6, kernel_size=(1, 1), activation="softmax"))
1386 #model2.add(layers.Dense(6, activation="softmax"))
1387 model13.summary()
1388
1389 #compile model using accuracy to measure model performance
1390 model13.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
1391
1392 train_ex = np.expand_dims(train_windows, axis=0)
1393 train_windows_r = train_windows.reshape(len(train_windows), 1, 800,7)
1394 val_windows_r = val_windows.reshape(len(val_windows), 1, 800,7)
1395
1396 # Train the model for 1 epoch from Numpy data
1397 batch_size = 64
1398 print("Fit on NumPy data")
1399 print(train_windows.shape)
1400 print(np.array(train_classes).shape)
1401 history13 = model13.fit(train_windows_r, np.array(train_classes).reshape(len(train_classes), 1, 1,6),
        ↪ validation_data=(val_windows_r, np.array(val_classes).reshape(len(val_classes), 1, 1,6)),batch_size=
        ↪ batch_size, epochs=25)
1402
1403 plt.plot(range(0,len(history13.history['val_accuracy'])), history13.history['val_accuracy'], label = "validation")
1404 plt.xlabel("Época")
1405 plt.ylabel("Accuracy")
1406 plt.title("Accuracy para cada época sin canal 1")
1407 plt.plot(range(0,len(history13.history['accuracy'])), history13.history['accuracy'], label = "train", color = 'purple')
1408 plt.legend()
1409
1410 plt.plot(range(0,len(history13.history['val_loss'])), history13.history['val_loss'], label = "validation")
1411 plt.xlabel("Época")
1412 plt.ylabel("Loss")
```

```
plt.title("Loss para cada época sin canal 1")
plt.plot(range(0,len(history13.history['loss'])), history13.history['loss'], label = "train", color = 'purple')
plt.legend()

y_pred = model13.predict(val_windows_r)
y_pred = np.argmax(y_pred, axis=3)
y_pred = y_pred.reshape(val_windows_r.shape[0],)

sns.heatmap(confusion_matrix(y,y_pred, normalize = 'all'), fmt='g')

"""## output"""

from typing_extensions import ParamSpecArgs
def test_dataset_generator_matrix2(df):
  #normalizacion
  for channel in range(1, 9):
    if channel != 8:
      all = df.iloc[:,(channel-1)*800+1 : channel*800 +1]
      all_normed = all/all.abs().max()
      df.iloc[:,(channel-1)*800+1 : channel*800 +1] = all_normed

  #i ----> fila
  #j ----> columna
  id=0
  #dataset sera una matriz de dimensiones: cantidad de ventanas x dimension ventana x cantidad canales
  w_test_df = np.zeros((len(df), 800, 7))
  condition = True
  while condition:
    if len(df) <= id:
      condition = False
    else:
      for channel in range(1,9):
        if channel != 8:
          window = df.iloc[id,(channel-1)*800 + 1:channel*800 +1].to_numpy()
          for i in range(0,len(window)):
            if channel < 8:
              w_test_df[id][i][channel-1] = window[i]
            else:
              w_test_df[id][i][channel-2] = window[i]
      id += 1

  return w_test_df
test_windows = test_dataset_generator_matrix2(test_df)

predictions = model13.predict(test_windows.reshape(len(test_windows), 1, 800,7))
p = np.argmax(predictions, axis=3)
p = p.reshape(672,)

r = pd.DataFrame(columns = ['Id', 'Category'])
for i in range(0,len(p)):
  r.loc[i,'Id']=i
  r.loc[i,'Category']=p[i]+1

compression_opts = dict(method='zip',archive_name='signal8.csv')

r.to_csv('signal8.zip', index=False, compression=compression_opts)
```