Universidad Nacional Autónoma de México

Facultad de ingeniería

Ingeniería en computación

Laboratorio de sistemas de comunicaciones

Reporte de prototipo

Gakki (楽器)

Profesor: **ING. CHRISTIAN HERNÁNDEZ SANTIAGO**

Grupo laboratorio: **11** *Grupo teoría*: **5**

Alumno: **Aguilera Valderrama Alexis Fernando**
No. cuenta: **316217701**

Fecha: **1 / Diciembre / 2021**

Semestre: **2022-1**

Objetivo

Usando las ventajas que nos ofrece la transformada de fourier, elaborar una red neuronal que sea capaz de analizar audios de instrumentos para determinar de cuál se trata.

Elaborar una GUI para que el uso de la red neuronal sea mas amigable.

Introducción

La serie de Fourier

Esta serie se usa en análisis de señales para representar las componentes senoidales de una onda periódica no senoidal, es decir, para cambiar una señal en el dominio del tiempo a una señal en el dominio de la frecuencia. En general, se puede obtener una serie de Fourier para cualquier función periódica, en forma de una serie de funciones trigonométricas con la siguiente forma matemática.[1]

$$f(t) = A_0 + A_1 \cos\alpha + A_2 \cos 2\alpha + A_3 \cos 3\alpha + \cdots + A_n \cos n\alpha + B_1 \operatorname{sen}\beta + B_2 \operatorname{sen} 2\beta + B_3 \operatorname{sen} 3\beta + \cdots + B_n \operatorname{sen} n\beta$$

donde $\alpha = \beta$

Figure 1: Expresión para la serie de Fourier

Otra forma de expresarla es con:

$$\frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos \frac{2n\pi}{T} t + b_n \sin \frac{2n\pi}{T} t \right]$$

Figure 2: Serie de fourier
https://es.wikipedia.org/wiki/Serie_de_Fourier

Muchas formas de onda que se manejan en los sistemas normales de comunicaciones no se pueden definir en forma satisfactoria con ecuaciones matemáticas; sin embargo, es de interés primordial su comportamiento en el dominio de la frecuencia. Con frecuencia hay necesidad de obtener este comportamiento de señales que se captan en el dominio del tiempo, es decir, en tiempo real.[1]

Ésta es la razón por la que se desarrolló la transformada discreta de Fourier. En esa transformación se muestrea una señal en el dominio del tiempo, en tiempos discretos. Las muestras se alimentan a una computadora donde un algoritmo calcula la transformación. Sin embargo, el tiempo de computación es proporcional a n^2, siendo n la cantidad de muestras.[1]

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

Figure 3: Expresión de Transformada discreta de fourier (https://es.wikipedia.org/wiki/Transformada_de_Fourier_discreta)

En consecuencia se desarrolló, en 1965, un nuevo algoritmo, llamado transformada rápida de Fourier(FFT, de fast Fourier transform), por Cooley y Tukey. Con la FFT, el tiempo de cómputo es proporcional a n $\log_2$ n[1]

Red neuronal

Las redes neuronales artificiales (ANN) están compuestas por capas de nodos, que contienen una capa de entrada, una o más capas ocultas, y una capa de salida. Cada nodo, o neurona artificial, se conecta a otro y tiene un peso y un umbral asociados. Si la salida de cualquier nodo individual está por encima del valor de umbral especificado, dicho nodo se activa, enviando datos a la siguiente capa de la red. De lo contrario, no se pasan datos a la siguiente capa de la red.[2]

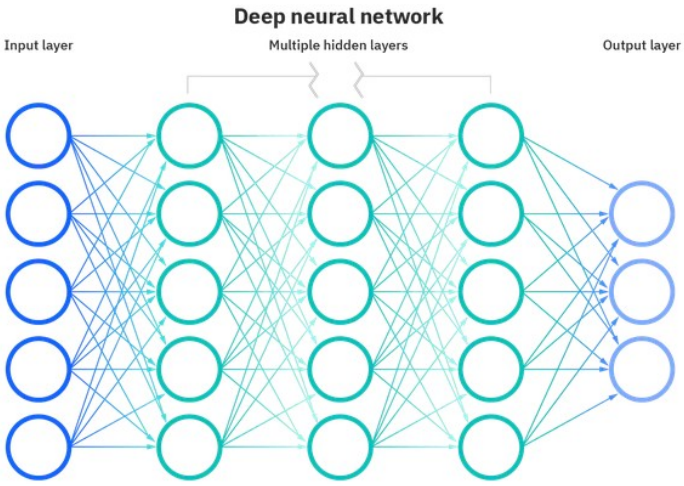

Figure 4: Red neuronal (https://www.ibm.com/mx-es/cloud/learn/neural-networks)

Para fines de mejorar la red se utilizan dos artefactos muy útiles los cuales son la función de costos y el optimizador.

La función de costos (perdida – loss) sirve para calcular qué tanto se alejó la predicción realizada del dato real esperado. Por su parte, el optimizador usa la información brindada por

la función de costos para modificar los parámetros de cada neurona, para que así se obtengan mejores resultados.

Python

Python es un lenguaje de programación interpretado ampliamente usando en el campo del machine learning y deep learning, esto por que muchas organizaciones han generado librerías para trabajar con algoritmos de estos campos, dichas librerías son sklearn, tensorflow y pytorch principalmente.

Además también tiene la librería scipy que implementa el algoritmo de la transformada rápida de fourier (FFT).

Entonces para elaborar el software, haremos uso de python con la librería pytorch para implementar la red neuronal y también de la librería scipy para poder obtener transformadas de fourier de audios de instrumentos. Para la GUI se hará uso de tkinter, el cual tambien forma parte de las librerías de python.

Bibliografía

[1] Tomasi, W., Hernández, G. M., & Pozo, V. G. (2003). Sistemas de comunicaciones electrónicas. Pearson Educación.

[2] Redes neuronales. (2021, 14 septiembre). IBM. https://www.ibm.com/mx-es/cloud/learn/neural-networks

Conjunto de datos

Como es bien sabido, toda inteligencia artificial necesita conseguir datos para aprender.

Para este proyecto se hizo una búsqueda en youtube de videos largos que contengan piezas tocadas sólo por el instrumento de interés. Como ya se mencionó estos son 8: Piano, flauta, shamisen, bajo, violin, kalimba, saxofón y guitarra. Los videos utilizados fueron los siguientes:

Bajo: https://www.youtube.com/watch?v=oRzyceprYYQ&t=10920s (1era Hora)

Flauta: https://www.youtube.com/watch?v=xFECNsk5Fo0&t=6044s

Shamisen: https://www.youtube.com/watch?v=8HTNaqEMOgk&t=1177s

Piano: https://www.youtube.com/watch?v=JPZ-QyVBOf8&t=1321s

Piano: https://www.youtube.com/watch?v=-gwyHUPvHbc

Violin: https://www.youtube.com/watch?v=i2mTmi9tUag&t=2873s

Kalimba: https://www.youtube.com/watch?v=g9zvwFbu-UY&t=2572s

Guitarra: https://www.youtube.com/watch?v=JpzIKEp5GxY&t=971s

Saxofón: https://www.youtube.com/watch?v=89A0vvoeI6M

Estos audios de guardaron todos en una carpeta llamada samples

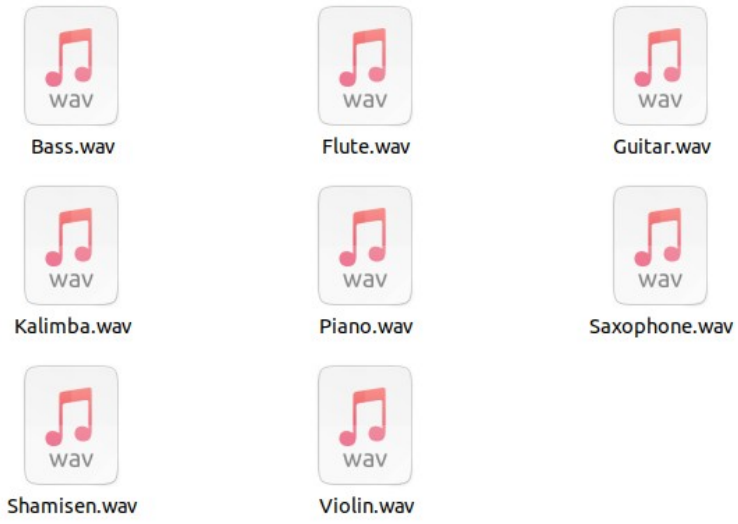

Figure 5: Fuente de datos

Cabe mencionar que todas las pistas tienen una frecuencia de muestreo de 48000Hz,

Implementación

La implementación de la inteligencia se divide en tres scripts de Python: Gakki, Core y red_neuronal, estos se irán explicando a continuación.

Red_neuronal.py

Define la red neuronal a utilizar para hacer las predicciones de audio.

Sus características principales son:

- Una capa de input de tamaño correspondiente a los puntos totales de la transformada de fourier a generar, la cual se explicará más adelante.
- Una capa escondida de 120 neuronas con función de activación ReLu (Rectificador linea unitario).
- Otra capa escondida de 70 neuronas igualmente con Relu.
- Una última capa de output con un número de neuronas correspondiente a la cantidad de instrumentos a predecir, en este casos son 8.
- Tiene una optimizador de tipo Adam con un learning rate de 0.001.
- Tiene una función de costes tipo smooth_l1_loss.
- Se aplica una función softmax para obtener predicciones probabilísticas.

El método process es para entrenar a la red mientras que la de predict ya para usar a la red en sí.

Core.py

Este script define funciones útiles para poder procesar los datos y conseguir transformadas de Fourier apartir de la pistas descargadas.

Se definen las siguientes variables para el cálculo de la transformada de Fourier

```
segundos = 0.9
umbral = 10
sample_rate = 48000

chunk = int(segundos * sample_rate)

tam = int(segundos * sample_rate / 2)

modelo = Gakki_NN(tam, 8)
```

Figure 6: Variables para calcular la variable de fourier

Lo que nos dicen estas variables es que por cada 0.9 segundos en cada pista se irá generando una transformada de fourier, la cuál va a tener tamaño segundos * sample_rate / 2, ya que es unilateral. **Es decir, que con estos parámetros, el tamaño de la capa de input de la red neuronal será de 0.9 * 48000 / 2 = 21600 neuronas en la capa de entrada, en otras palabras, cada transformada de fourier tendrá 21000 puntos.**

Adicionalmente la variables umbral sirve para aplicarle un limpiado a las pistas, de tal forma que todos los samples que quedan por debajo de éste umbral son eliminadas

Método CreateDataSet()

El método se encarga de procesar cada pista de audio obteniendo una transformada cada 0.9 segundos **al canal izquierdo de la pista**. Entonces por cada pista se guardan aproximadamente 4000 transformadas de fourier (demasiadas).

El pedazo de código encargado de obtener la transformada de fourier es la siguiente:

```
samples = mono[chunk*i:chunk*(i + 1)]

if len(samples) < chunk:
    break

transformada = np.abs(fft(samples) * 2 / (256 * chunk))

transformada = transformada[int(chunk/2):len(transformada)]

for value in transformada:
    f.write(str(value) + "\n")

f.write("\n")

i = i + 1
```

Figure 7: Calculo de la transormada

Visualizando las transformadas de Fourier se obtiene lo siguiente:

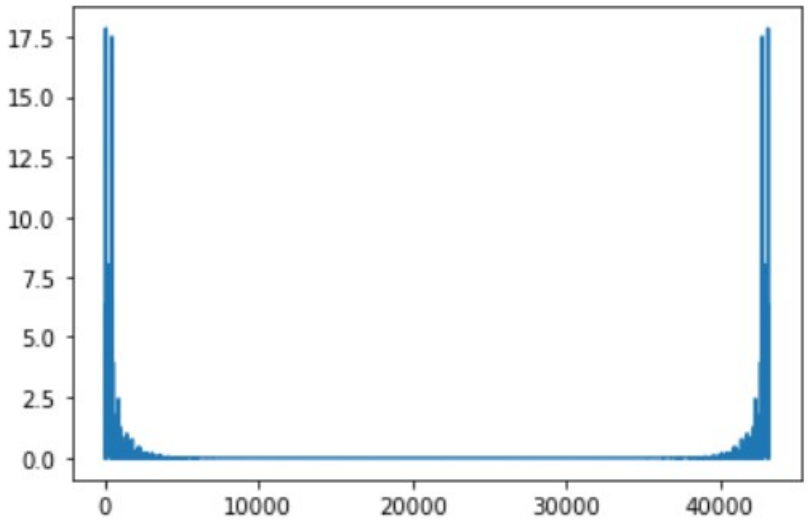

Figure 8: Tranforamada de fourier bilateral de piano

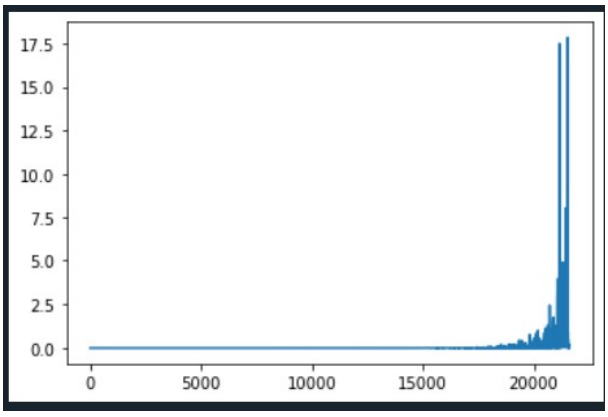

Figure 9: Transformada unilateral escalada de Fourier de piano

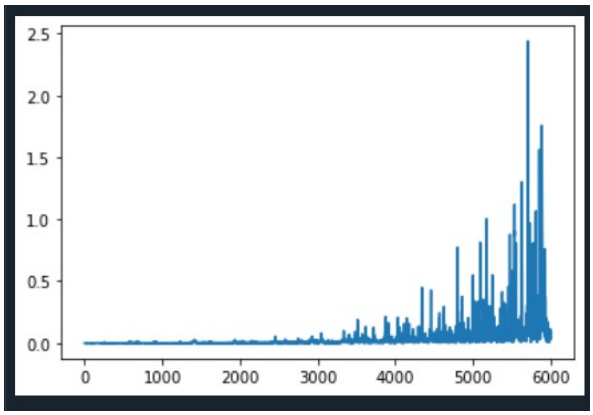

Figure 10: Transformada unilateral escalada de fouerier (Zoom) de piano

En este caso la transformada que recibe la red neuronal es la que se muestra en la figura 5. SI bien hay muchas muestras donde el valor es casi 0, al intentar quitar estos puntos, la predicción empeoraba por lo tanto se decidió dejarlas.

Haciendo el proceso de cada transformada se producieron archivos .txt con las transformadas representadas en números

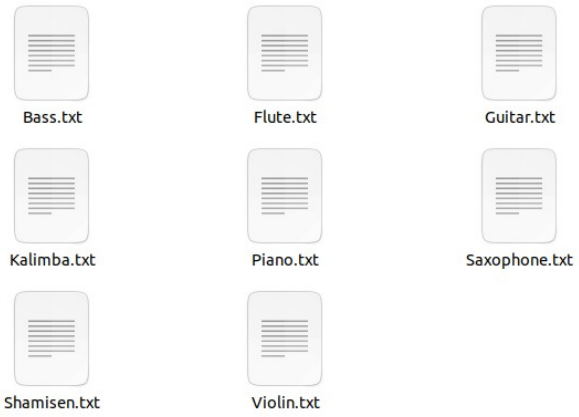

Figure 11: Dataset de transformadas de fourier de cada instrumento

Como se puede esperar estos archivos tienen demasiados datos y son muy pesados, entre los 8 documentos se producian 15.5 GB de datos

```
78501 0.0006549335951985646
78502 0.0003793613584489632
78503 0.0005929037974511885
78504 0.00013582835806938821
78505 0.0004723390907018966
78506 0.00020568206783580772
78507 0.0006350068701328133
78508 0.0003788115569710028
78509 0.0009784313991598262
78510 9.161415745444597e-05
78511 0.00010546047586222966
78512 1.976532314059585e-05
78513 0.00033050210844133056
78514 0.00039105419932027166
78515 0.0007077087956972812
78516 0.00036973229727482016
78517 0.00028722550041799074
78518 0.0007455261764431414
78519 0.0002777434841194109
78520 0.0004311295732722491
78521 0.0006041112687380638
78522 0.0005054655414222103
78523 0.00041804127404999143
78524 0.0003461387777522939
78525 0.00039183056940060714
```

Figure 12: Ejemplo de transformada guardada en txt

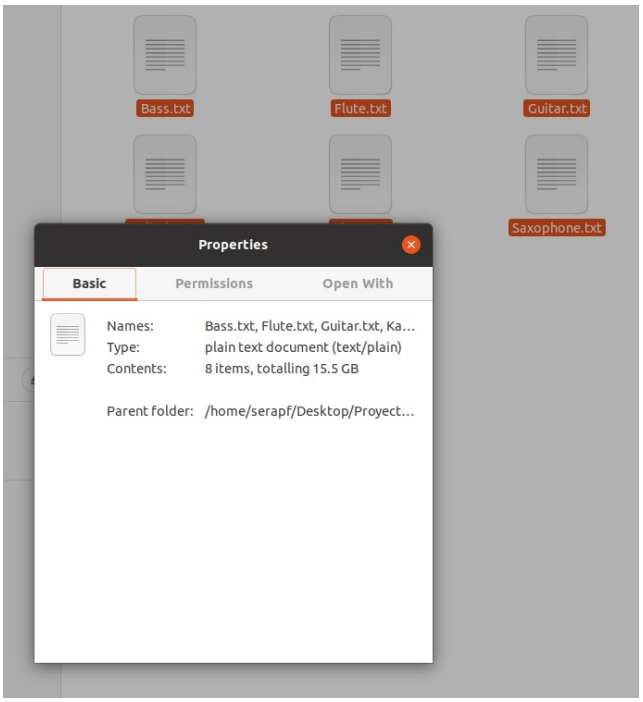

Figure 13: Tamaño total de datos

Entrenamiento

El método encargado de entrenar a la red se llama train(). El funcionamiento del entrenamiento se basa en lo siguiente

```
batch = []

samples_per_instrument = 30

instrument_dict = {
        "Bass"     : 0,
        "Flute"    : 1,
        "Guitar"   : 2,
        "Kalimba"  : 3,
        "Piano"    : 4,
        "Saxophone": 5,
        "Shamisen" : 6,
        "Violin"   : 7,
    }

prob_instrument = {
        0    : [1.0,0,0,0,0,0,0,0], ##Bajo
        1    : [0,1.0,0,0,0,0,0,0], ##Flauta
        2    : [0,0,1.0,0,0,0,0,0], ##Guitarra
        3    : [0,0,0,1.0,0,0,0,0], ##Kalimba
        4    : [0,0,0,0,1.0,0,0,0], ##Piano
        5    : [0,0,0,0,0,1.0,0,0], ##Saxophone
        6    : [0,0,0,0,0,0,1.0,0], ##Shamisen
        7    : [0,0,0,0,0,0,0,1.0], ##Violin
    }

err = [0,0,0,0,0,0,0,0]

tam = int(segundos * sample_rate / 2)

modelo = Gakki_NN(tam, 8)
```

Figure 14: Variables para entrenamiento

Como lo archivos son muy grandes, no es posible cargar todos en memoria principal, por lo tanto se harán en conjuntos (batches) de 30 elementos por instrumento, es decir que por cada batch se cargaran 30 transformadas de cada instrumentos (30 * 8 = 240 transformadas por batch).

Otro aspecto importante es el diccionario prob_instrument, ya que éste tendrá el vector esperado por cada instrumento. En otras palabras, le funcionará a la red neuronal para determinar que tan bien hizo la predicción.

Adicionalmente se crea la red con un tamaño de la capa de input igual al tamaño de cada transformada que esta muy bien definida, y con tamaña de capa de salida igual a la cantidad de instrumentos a predecir.

Ahora se abren todos los documentos y con el método getsamples() se consiguen las transformadas de cada documento y se etiquetan con el respectivo instrumento.

```
i = 1

with open("./Datasets/Bass.txt") as bass_file,\
     open("./Datasets/Flute.txt") as flute_file,\
     open("./Datasets/Guitar.txt") as guitar_file,\
     open("./Datasets/Kalimba.txt") as kalimba_file,\
     open("./Datasets/Piano.txt") as piano_file,\
     open("./Datasets/Saxophone.txt") as saxophone_file,\
     open("./Datasets/Shamisen.txt") as shamisen_file,\
     open("./Datasets/Violin.txt") as violin_file:

        while True:

            getSamples(batch, bass_file,     samples_per_instrument,instrument_dict["Bass"])
            getSamples(batch, flute_file,    samples_per_instrument,instrument_dict["Flute"])
            getSamples(batch, guitar_file,   samples_per_instrument,instrument_dict["Guitar"])
            getSamples(batch, kalimba_file,  samples_per_instrument,instrument_dict["Kalimba"])
            getSamples(batch, piano_file,    samples_per_instrument,instrument_dict["Piano"])
            getSamples(batch, saxophone_file, samples_per_instrument,instrument_dict["Saxophone"])
            getSamples(batch, shamisen_file, samples_per_instrument,instrument_dict["Shamisen"])
            getSamples(batch, violin_file,   samples_per_instrument,instrument_dict["Violin"])

            if i == 90:
                break;

            random.shuffle(batch)

            accumulated_loss = []
```

Figure 15: Apertura de documentos y estracción de datos por batch

El break cuando se llega al batch 90 (i == 90), solo es un limite para que no tarde mucho en entrenar, si se quita, recorrerá todas las transformadas generadas.

En las siguiente linea de código se va iterando cada elemento del batch, o sea, cada transformada etiquetada con su instrumento y se le pasa al método process del modelo creado. El método anterior regresa la respuesta de la predicción y una perdida generada que determina cuanto error hubo en la predicción, la cual es añadida a un error acumulado del batch. Adicionalmente si la predicción se hizo de manera errónea, en una lista que cuenta los errores por instrumento se suma en uno el contador al instrumento correspondiente.

```
for element in batch:
    loss,prob = modelo.process(element[0], prob_instrument[element[1]])
    accumulated_loss.append(loss)

    prob = prob.detach().numpy()[0].tolist()

    max_val = max(prob)

    index = prob.index(max_val)

    if index != element[1]:
        err[element[1]] = err[element[1]] + 1



print("-----------------------------------------------")
print("Batch no. ", i)
print("Current samples num ", samples_per_instrument * i)
print("Batch Size ", len(batch)/8)
print("Batch loss ", statistics.mean(accumulated_loss))
print("Errores ", err)
print("-----------------------------------------------\n")
batch = []
i = i + 1
```

Figure 16: Entrenamiento del modelo por batch

Al final cuando se terminan los elementos del batch se muestra la media del error acumulado y el estado actual de la lista de errores.

```
------------------------------------------------
Batch no.  1
Current samples num  30
Batch Size  30.0
Batch loss  0.03472679077292216
Errores  [1, 27, 7, 7, 12, 19, 8, 19]
------------------------------------------------

------------------------------------------------
Batch no.  2
Current samples num  60
Batch Size  30.0
Batch loss  0.015198223657603122
Errores  [2, 35, 25, 8, 13, 19, 10, 22]
------------------------------------------------

------------------------------------------------
Batch no.  3
Current samples num  90
Batch Size  30.0
Batch loss  0.017103463510319637
Errores  [7, 43, 38, 8, 14, 20, 11, 29]
------------------------------------------------
```

Figure 18: Reporte de desempeño a iteraciones bajas

```
Current samples num  2550
Batch Size  30.0
Batch loss  0.0016351685774677954
Errores  [275, 154, 284, 111, 141, 201, 107, 130]
------------------------------------------------

------------------------------------------------
Batch no.  86
Current samples num  2580
Batch Size  30.0
Batch loss  0.0044487220465235
Errores  [275, 154, 284, 111, 144, 207, 107, 132]
------------------------------------------------

------------------------------------------------
Batch no.  87
Current samples num  2610
Batch Size  30.0
Batch loss  0.0028761072991324906
Errores  [275, 155, 285, 112, 144, 209, 107, 134]
------------------------------------------------

------------------------------------------------
Batch no.  88
Current samples num  2640
Batch Size  30.0
Batch loss  0.01119140095181157
Errores  [275, 158, 293, 114, 152, 212, 108, 134]
------------------------------------------------

------------------------------------------------
Batch no.  89
Current samples num  2670
Batch Size  30.0
Batch loss  0.004737559977251955
Errores  [275, 158, 295, 114, 152, 220, 108, 134]
------------------------------------------------
```

Figure 17: Reporte de desempeño a iteraciones altas

El error acumulado es de 0, si todas las predicciones del batch fueron exitosas.

En el entrenamiento el instrumento que mejor tuvo desempeño el shamisen (108 errores de 2670 muestras) y el de peor fue la guitarra (295 errores de 2670 muestras); sin embargo esto no significa que la guitarra será la peor en ser identificada; ya que si bien pudo tener más errores, pudo también haber aprendido a identificar más tonos y timbres de la guitarra.

Con el modelo entrenado se procede a guardarlo para poder usarlo en futuras sesiones.

Figure 19: Red neuronal generada

Testing

Para poner a prueba el modelo generado se puede ejecutar el método Listen(). Lo que hace el método es que graba un audio con frecuencia de muestreo y tiempo iguales a los que se utilizaron para dividir las pistas de audio en el conjunto de datos, en este caso fue de 0.9 Hz a 48000 Hz.

Teniendo la pista grabada, se procede a aplicar la transformada de fourier exactamente igual a como se ha venido haciendo.

```
def Listen():

    record()

    samplerate, data = wavfile.read("recording1.wav")

    mono = data[:,0]

    samples = mono[0:chunk]

    transformada = np.abs(fft(samples) * 2 / (256 * chunk))

    transformada = transformada[int(chunk/2):len(transformada)]

    pred = modelo.predict(transformada).detach().numpy()[0].tolist()

    max_val = max(pred)

    index = pred.index(max_val)

    print(ins[index])

    return ins[index]
```

Figure 20: Creación de audio y espectro frecuencial de la pista de prueba

Metiendo este método a un loop infinito podemos generar predicciones cada 0.9 segundos.

```
Guitar
/home/serapf/Desktop/Github/Proyecto_ST_Gakki/red_neuronal.py:66: UserWarning:
volatile was removed and now has no effect. Use `with torch.no_grad():` instead
  pred = self.model(Variable(chunkdata, volatile = True))*100
/home/serapf/Desktop/Github/Proyecto_ST_Gakki/red_neuronal.py:67: UserWarning:
Implicit dimension choice for softmax has been deprecated. Change the call to
include dim=X as an argument.
  probs = F.softmax(pred)
Guitar
Guitar
Guitar
Guitar
Guitar
Guitar
Guitar
Guitar
Guitar
Guitar
Saxophone
Saxophone
Saxophone
Guitar
Guitar
Guitar
Saxophone
```

GUI

Gakki.py es el script que define la GUI, como se mencionó esta hecha en tkinter, donde se definen un cuadro de imagen donde de mostrará el instrumento de la predicción, un botón para comenzar las predicciones y cuadros de texto con información extra.

Este módulo le dira a core que cargue el documento last_brain.pth para cargar la red neuronal.

```
import Core

Core.load_model()
```

Figure 21: Carga del modleo guardado

EL botón para “comenzar” tendrá asociado un método llamado Comenzar() el cuál creará un hilo que correrá paralelamente el método Ejecutar(). Este último se encarga de ejecutar el método Listen() de Core.py y con la respuesta, carga la imagen necesaria.

```
def Comenzar():
    global comenzar
    if comenzar == False:
        thread = Thread(target = Ejecutar)
        thread.start()
        comenzar = True
    else:
        print("Ya esta corriendo")

def Ejecutar():
    my_label.config(text = "Escuchando...")
    while True:
        respuesta = Core.Listen()

        imagen = ImageTk.PhotoImage(Image.open(respuesta + ".jpg").resize((500,300)))
        my_image.config(image = imagen)
        my_label.config(text = respuesta)
```

Figure 22: Método para correr la red neuronal

Ejecución

Para poder ejecutar Gakki en cualquier computadora se debe tener python instalado en la computadora y algunas librerías, para la instalación de ellas se puede usar el instalador de paquetes pip. Por lo tanto, instalando las librerías desde una consola, ya sea en linux o Windows, se tendría:

- pip install pillow
- pip install matplotlib
- pip install scipy
- pip install sounddevice
- pip install wavio
- pip install torch
- pip install torchvision

Estas dos últimas dos librerías son las que definen a la inteligencias artificial, por lo tanto pesarán más que las otras.

Algunas librerías ya podrían venir instaladas dependiendo cómo se instale python.

Una vez instaladas todas la librerías, se procede a ejecutar el script “Gakki.py”. Para ejecutar el script se debe abrir una consola en el directorio donde esté el script y ejecutar tanto para Windows y Linux lo siguiente:

- python Gakki.py

Uso del programa

Con lo anterior se ejecutará el software apareciendo la siguiente imagen inicial:

Figure 23: Pantalla inicial de Gakki

Para usar el programa se tienen dos opciones:

- Usar un micrófono para captar sonidos
- Usar la salida stereo del sistema para captar audio, en otras palabras, obtener audio directo de la computadora.

Para en el caso del micrófono se tiene que ir a la configuración de audio correspondiente al sistema operativo y seleccionar el micrófono que se quiere utilizar.

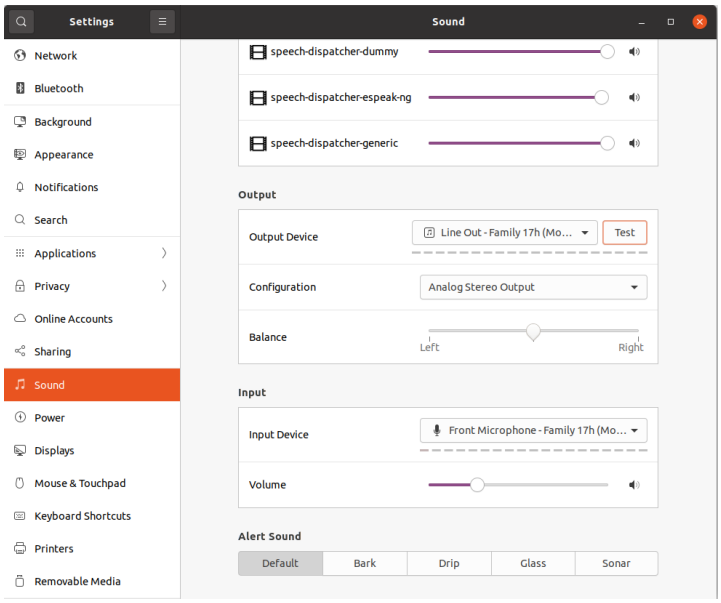

Figure 24: Ventana de dispositivos de audio de linux (Ubuntu)

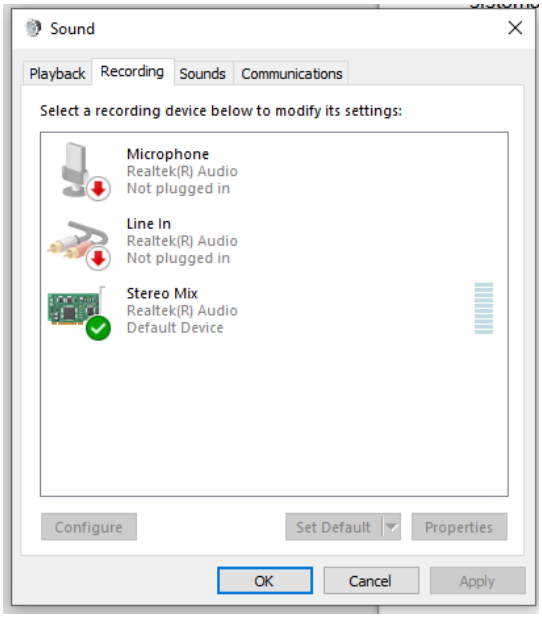

Figure 25: Ventana de dispositivos de audio de Windows

Para la salida stereo (la cual se recomienda que se use para tener un audio más limpio), se pueden seguir dos estrategias:

- Desconectar todos los micrófonos externos de la computadora para que deje disponible la entrada stereo.
- Seleccionar la entrada stereo desde la configuración del sistema operativo.

En Linux no se puede seguir esta estrategia, se debe de seguir la primera mencionada.

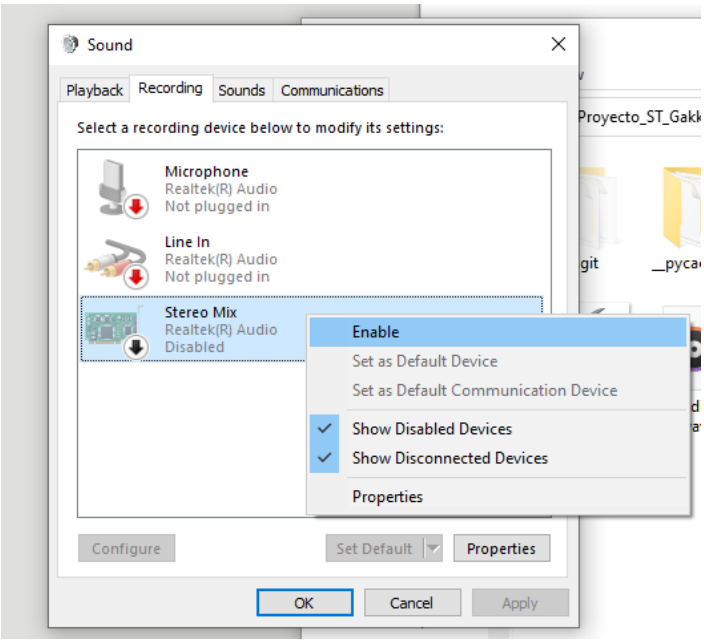

Figure 26: Activación de audio stereo de la computadora en Windows

Con la configuración deseada se puede dar click en “comenzar” para que la inteligencia artificial se ponga a escuchar y a predecir.

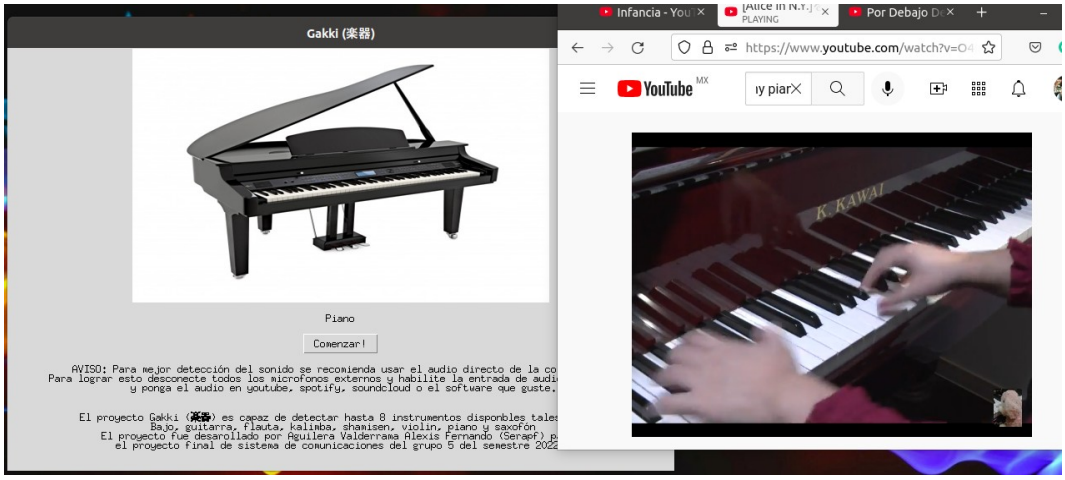

Figure 27: Probando Piano

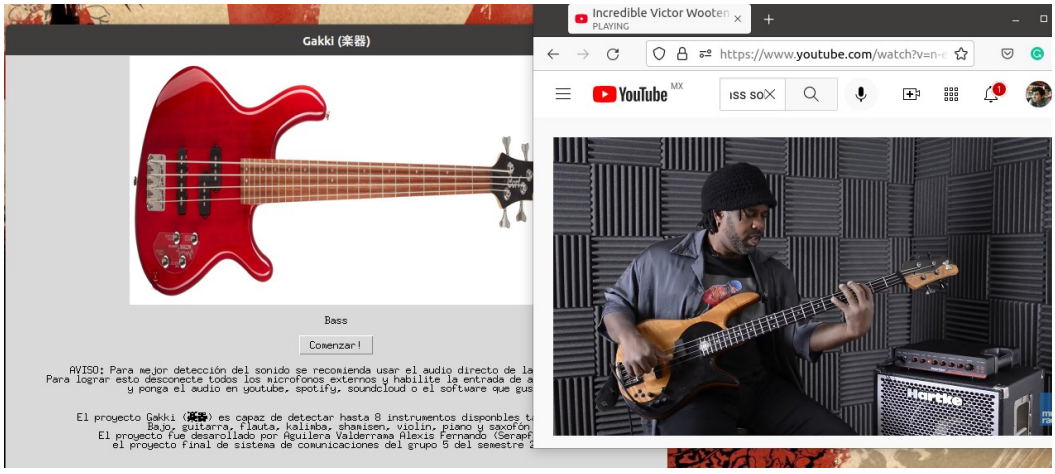

Figure 28: Probando bajo

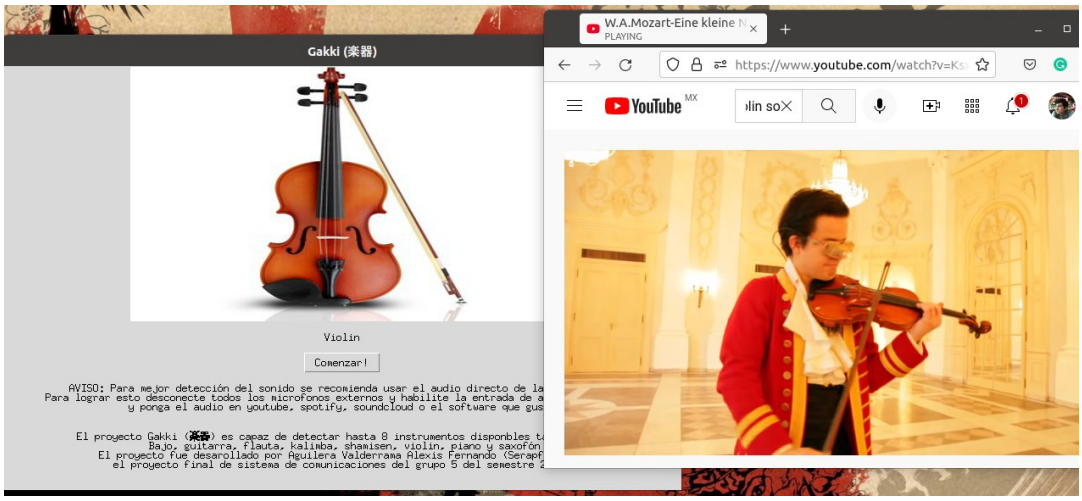

Figure 29: Probando Violin

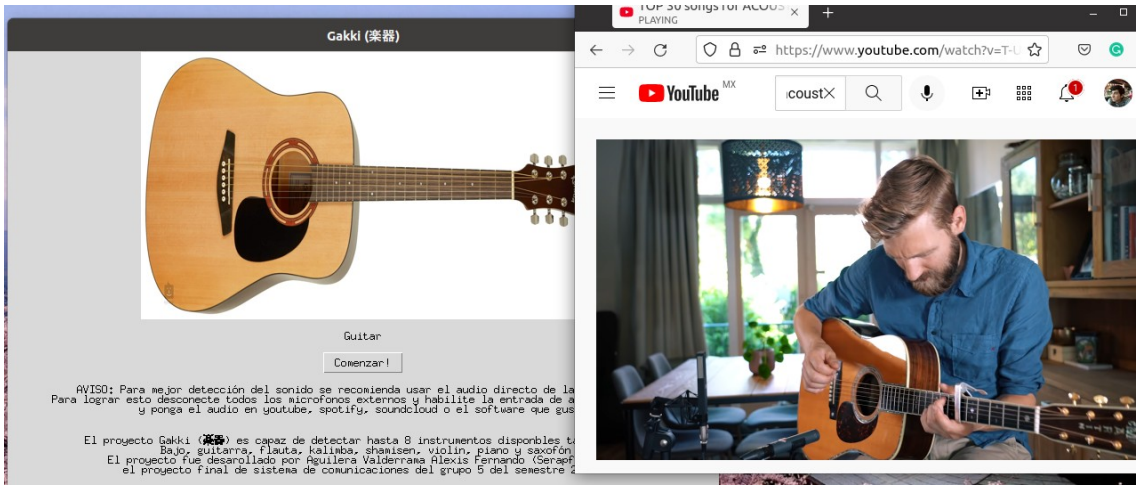

Figure 30: Probando guitarra

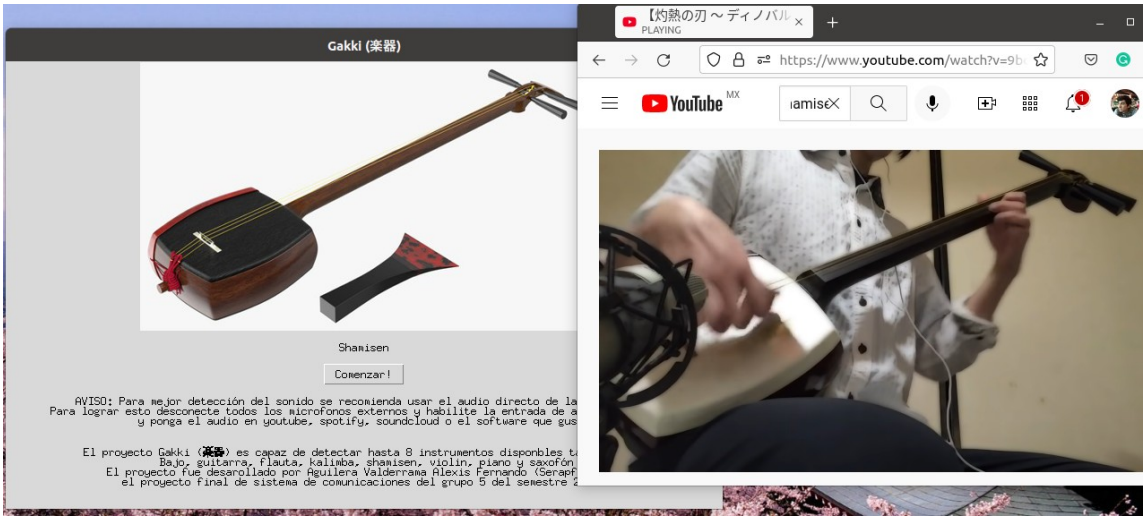

Figure 31: Probando shamisen

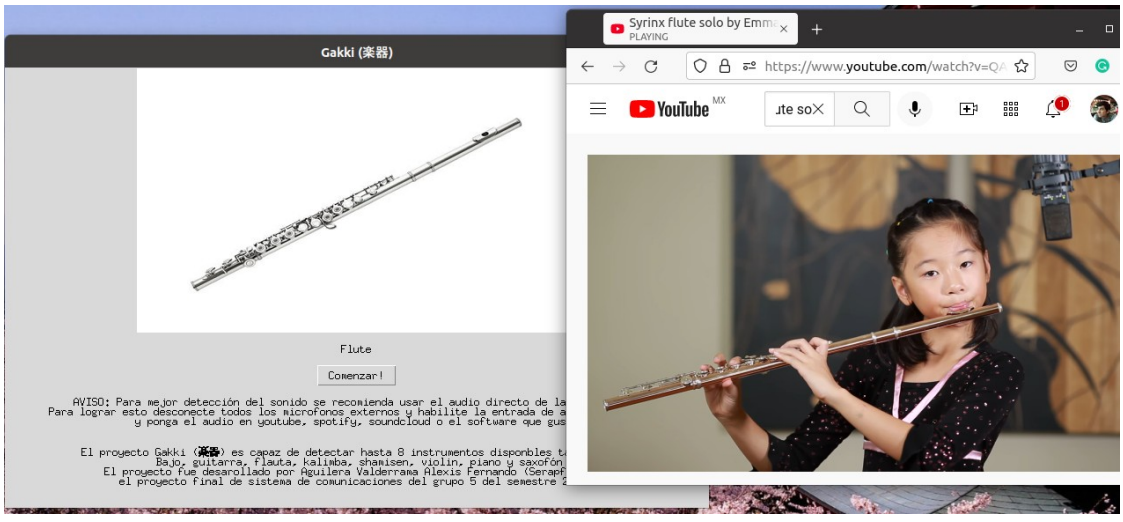

Figure 32: Probando Flauta

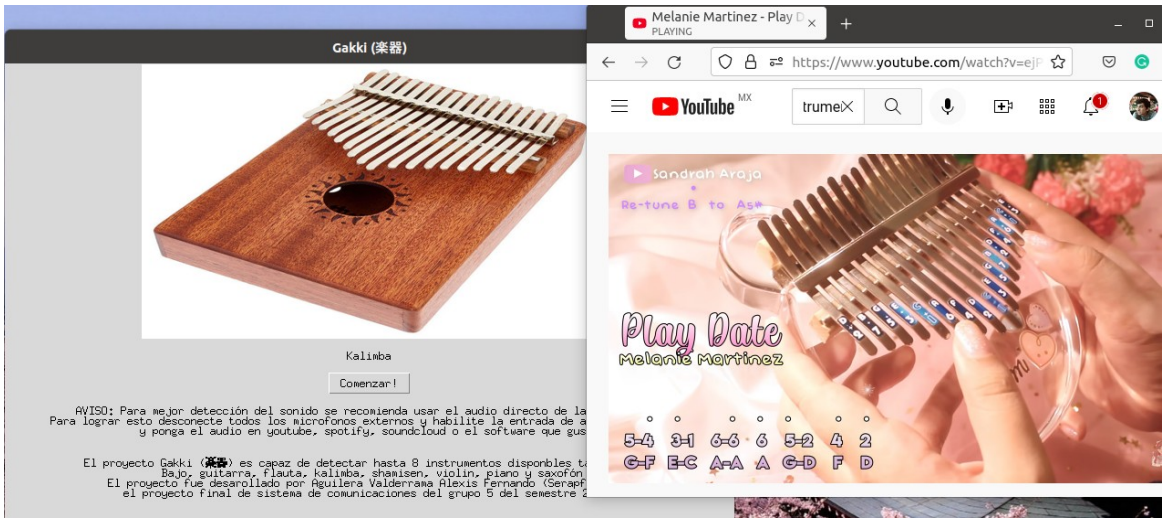

Figure 33: Probando Kalimba

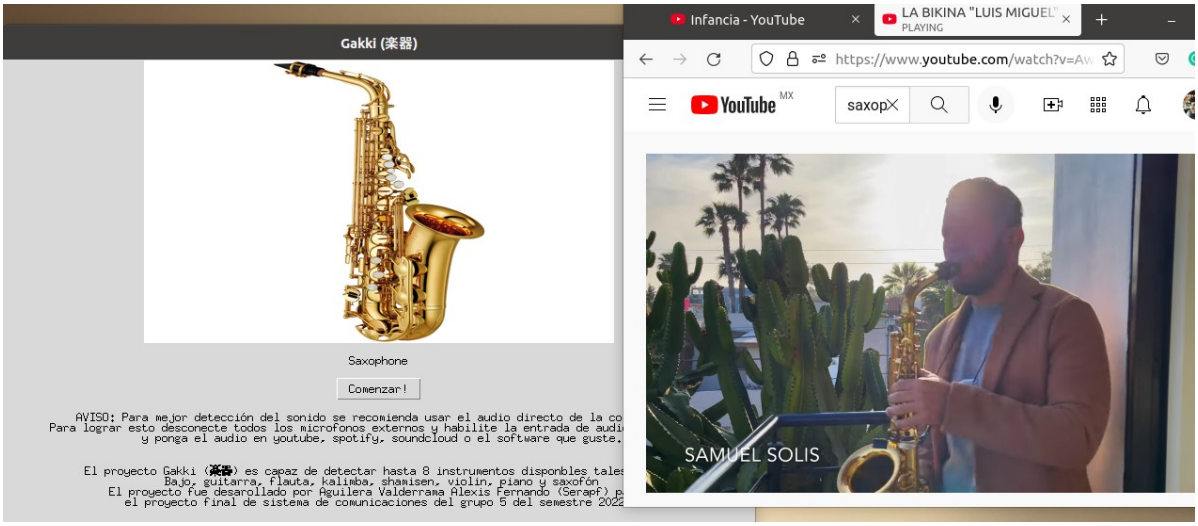

Figure 34: Probando saxofón

Conclusiones

Realizando el prototipo pude entender con mejor claridad en lo que consiste la transformada de Fourier, su representación e interpretación gráfica. Pero lo que más recapacite es sobre la utilidad que tiene la transformada. Si bien antes ya había tratando la transformada para ciertos fines no tan específicos, con este proyecto es la primera aplicación que hice con ella; aparte de también es mi primera aplicación propia usando redes neuronales (área que quiero profundizar en mis estudios).

Con respecto al desempeño de la red neuronal, se obtuvieron resultados muy satisfactorios para los alcances de este proyecto. El mejor caso para identificar a un instrumento es cuando solo se escucha al mismo en una pista de audio y el peor caso es cuando hay muchos instrumentos mezclados o se intenta detectar un instrumento que no es capaz de identificar.

Se intentaron muchas cosas para la mejora de la calidad de predicción, a continuación se mencionan los aspectos que se intentaron:

- Variar el tiempo de división de las pistas para la generación del dataset. Esto con el fin de tener notas de los instrumentos más atómicas. Sin embargo, no descarto la posibilidad de generar un algoritmo capaz de encanillar perfectamente cada nota del audio para tener mejores muestras.
- Variar el parámetro umbral para quitar muestras de amplitud muy pequeña.
- Cambiar la cantidad de capas y/o neuronas por cada capa con diferentes funciones de activación.
- Escoger optimizador y/o función de costos diferentes.
- Variar el learning rate del optimizador
- Descargar videos con audios con instrumentos más aislados y con mejor calidad.
- Reducir el tamaño de cada transformada cortando los primeros n-elementos ya que eran casi nulos.

Entonces la configuración marcada en el código es la mejor que pude obtener.

No obstante, hay métodos más complejos para poder realizar la identificación del instrumento. Una mejora muy directa a esta implementación es cambiar la red neuronal convencional por otra llamada red neuronal recurrente con memoria LSTM. Puedo hipotizar que el uso de este tipo de red podría mejorar con creces el desempeño, al grado que ni siquiera sería necesario calcular la transformada de fourier, se podría hacer simplemente ingresando el audio en el dominio del tiempo. Sin embargo también sería interesante ver los resultados si se le pasan las transformadas de fourier a la red recurrente LSTM.