1st Summer School Physical Engineering UNI 2022-Lima,Perú

Reporte Curso de Verano: Deep Learning

José Andrés Evangelio Vergaray[1]

[1]Facultad de Ciencias Naturales y Matemática, Universidad Nacional del Callao

March 8, 2022

Resumen

En este presente trabajo se expone la aplicación de lo aprendido en el curso de Verano: Deep Learning, organizado por el AEPIF - UNI. Se genera una red neuronal de 6 capas, para el reconocer patrones de voz y poder identificar el texto leído dentro de cada audio. En cada capa, se trabajo como una red convolucional cada capa para evitar conseguir valores muy pequeños dentro de los valores de gradiente al optimizar y poder aprovechar la capacidad de todas las capas.

Palabras Clave
Deep Learning, detección de mensajes, PyTorch, Redes Convolucionales

1 Metodología

Usaremos un conjunto de archivos para entrenar nuestra red neuronal en la detección de mensajes dentro de un audio. Para ello usaremos la información que genera un archivo de audio, transformamos la señal de intensidades por un conjunto de frecuencias, mediante una Transformada de Fourier. Con la transformada de Fourier generamos espectrogramas de cada audio, figuras con las que trabajaremos para entrenar nuestras redes convolucionales en la detección de patrones y su relación que tiene con el mensaje que desea transmitir en cada archivo de audio.

1.1 Definición de Espectrograma

De acuerdo a [1], el espectrograma se puede definir como una gráfica de intensidad (generalmente en una escala logarítmica, como dB) de la magnitud de la transformada de Fourier de corto tiempo (STFT). El STFT es simplemente una secuencia de algoritmos para generar una Transformada de Fourier Discreta a un conjunto de datos con ventanas, donde generalmente se permite que las ventanas se superpongan en el tiempo. Es una representación importante de los datos de audio porque la audición humana se basa en una especie de espectrograma en tiempo real codificado por la cóclea del oído interno [2]. El espectrograma se ha utilizado ampliamente en el campo de la música por computadora como guía durante el desarrollo de algoritmos de síntesis de sonido. Cuando se trabaja con un modelo de síntesis apropiado, hacer coincidir el espectrograma a menudo corresponde a hacer coincidir el sonido extremadamente bien.

1.2 Redes Convolucionales

Son redes neuronales usadas para procesar imágenes y poder detectar o categorizar objetos, clasificar escenas o clasificar imágenes en general mediante convolución [3].

El término de convolución posee una definición principal de aspecto matemático, donde un operador matemático transforma dos funciones f y g en una tercera función que en cierto sentido representa la magnitud en la que se superponen f y una versión trasladada e invertida de g [*Fuente*: Wikipedia].

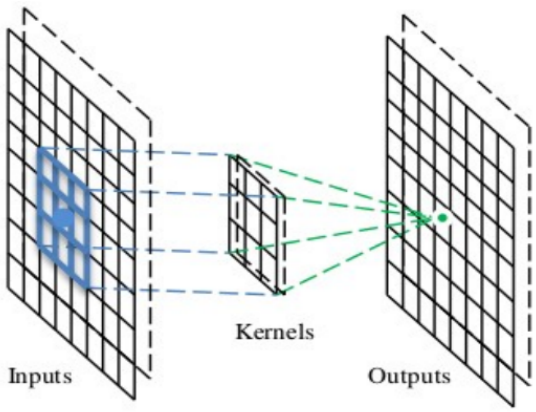

Figura 1: Representación de una imagen siendo procesada con una máscara o Kernel, para obtener Outputs resultantes. *Imagen obtenida de* [3].

La convolución en el procesamiento de imágenes consiste en filtrar una imagen usando máscaras, para obtener una imagen de salid que nos muestre patrones específicos acorde a las máscaras que usemos. Aquí cada píxel de salida es una combinación lineal

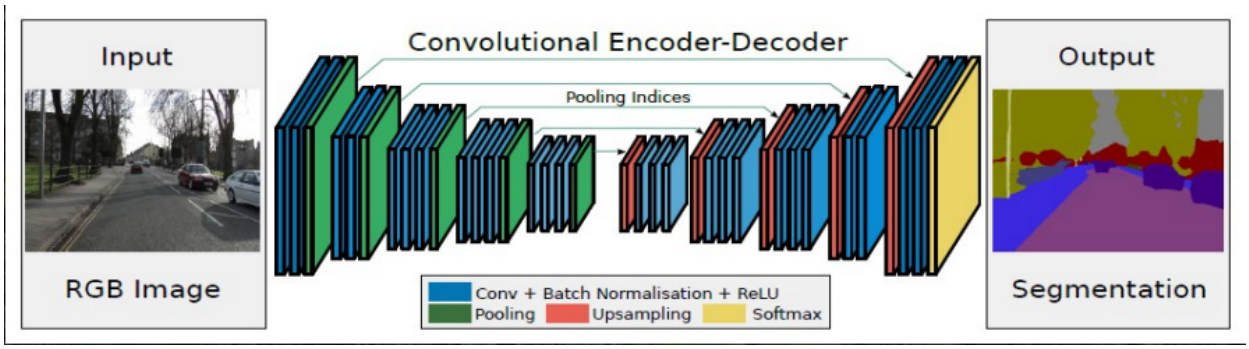

Figura 2: Esquema de procesamiento de imágenes mediante redes convolucionales. Los Conv. Kernel más el Batch Normalization, ReLu y Pooling representarán las primeras capas de la red neuronal. Los Upsampling y Softmax representarán las capas de Output o salida de la red neuronal. *Imagen obtenida de* [3].

de los píxeles de la imagen de entrada, como se muestra en la Figura 1. Aquí, se escoge un conjunto de píxeles que se operan matricialmente con un Kernel o matriz que funcionará como máscara, para obtener un conjunto de valores que reconstruirán una imagen.

Para una red convolucional se usará una serie de máscaras hasta obtener valores que la red pueda interpretar para optimizar cada capa de la red misma, disminuyendo la cantidad de píxeles en el proceso pero aumentando la cantidad de información que estas contienen. Al final, dependiendo de la función que queramos dar a la red convolucional, usará una segunda capa de máscaras que adecuará la información de entrada procesada y obtener un output acorde a nuestras necesidades, como se muestra en la Figura 2.

En particular, como esquema general se muestra un procesamiento de imágenes mediante redes convolucionales. Los cuadros en azul en cada capa indican el uso de máscaras de convolución, Batch Normalization y Rectificadores(ReLU). Los cuadros en verde representan las máscaras de pooling, para evitar que la información sea susceptible a la posición de las carácterísticas de la imagen. Los cuadros en rojo representan las capas de Upsampling, agregado para crear puntos de datos artificiales o duplicados o de la muestra de clase minoritaria para equilibrar la etiqueta de clase en la información procesada. Una última capa, cuadro amarillo, será agregado para darnos la data necesaria usando una función Softmax, que nos permite transformar un vector de números en vector de probabilidades.

1.3 Paquetería PyTorch

`PyTorch` es una biblioteca de tensores optimizada para el aprendizaje profundo utilizando GPU y CPU. Posee una gran cantidad de paqueterías que nos permiten generar redes neuronales con funciones específicas. `Torchaudio` es una paquetería para el procesamiento de audio y señal con `PyTorch`. Proporciona I/O, funciones de procesamiento de señales y datos, conjuntos de datos, implementaciones de modelos y componentes de aplicaciones.

Dentro de la red neuronal también usaremos `Conv1d`, el cuál aplica una convolución 1D sobre una señal de entrada compuesta por varios planos de entrada. En el caso más simple, el valor de salida de la capa con tamaño de entrada (N, C_{in}, L) y salida $(N, C_{\text{out}}, L_{\text{out}})$ se puede escribir como:

$$\begin{aligned}\text{out}\left(N_i, C_{\text{out}_j}\right) &= \text{bias}\left(C_{\text{out}_j}\right) \\ &+ \sum_{k=0}^{C_{\text{in}}-1} \text{weight}\left(C_{\text{out}_j}, k\right) \bigstar \text{input}\left(N_i, k\right)\end{aligned}$$

donde dónde $\bigstar$ es el operador de correlación cruzada válido, N es un tamaño de lote, C denota una serie de canales, L es una longitud de secuencia de señal.

2 Resultados

Trabajamos con 4 capas convolucionales, construidas a base de la paquetería `Conv1d` de `PyTorch`. Aquí, importaremos:

```
import torch
import torch.nn as nn
import torchvision
from scipy.io import wavfile
import torchaudio
from IPython.display import Audio, display
```

`torch` de `Pytorch`, al cuál llamaremos de ahora en adelante como `nn.` dentro del código. También usaremos `torchvision` para el procesamiento de imágenes, `scipy.io import wavfile`, `torchaudio` y `IPython.display import Audio, display`para poder trabajar los archivos de audio.

Cuando llamamos, dentro del código, a:

```
nn.Conv1d(in_channels,out_channels,kernel_size)
```

Estamos generando un números `in_channels` de canales de entrada para la imagen, produciendo una cantidad de `out_channels` canales producidos por la convolución, con un tamaño de núcleo convolusional igual a `kernel_size`. Usamos la paquetería para normalización por lotes unidimensional

`nn.BatchaNorm1d(num_features)`

con una cantidad de `num_features` tamaño de entrada esperado. Usaremos una función lineal de rectificación `ReLU` generando la operación en el lugar o `inplace=True`. Y terminamos aplicando una normalización por lotes con

`nn.BatchNorm1d(num_features)`

a través de una entrada 1D, siendo `num_features` el tamaño de entrada esperado. El código de la red convolucional completa será:

```
model = nn.Sequential(
    nn.Conv1d(1,32,100,bias=False),
    nn.BatchNorm1d(32),
    nn.ReLU(inplace=True),
    nn.MaxPool1d(4),
    nn.Conv1d(32,64,4,bias=False),
    nn.BatchNorm1d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool1d(4),
    nn.Conv1d(64,64,4,bias=False),
    nn.BatchNorm1d(64),
    nn.ReLU(inplace=True),
    nn.MaxPool1d(4),
    nn.Conv1d(64,128,4,bias=False),
    nn.BatchNorm1d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool1d(4),
    nn.Conv1d(128,128,4,bias=False),
    nn.BatchNorm1d(128),
    nn.ReLU(inplace=True),
    nn.MaxPool1d(4),
    nn.Conv1d(128,256,4,bias=False),
    nn.BatchNorm1d(256),
    nn.ReLU(inplace=True),
    nn.MaxPool1d(4),
    nn.Flatten(),
    nn.Linear(1536,248)
).to(dev)
```

Entrenamos nuestra red neuronal con un Dataset de 97 hablantes con 248 frases diferentes cada una. Los resultados del entrenamiento se pueden observar en las gráficas 3 y 4.

3 Discusión de resultados

Observando la curva de aprendizaje de la red neuronal y relacionando con la curva generada en la evaluación, podemos suponer que aumentar una mayor cantidad de iteraciones o épocas no genere un aumento

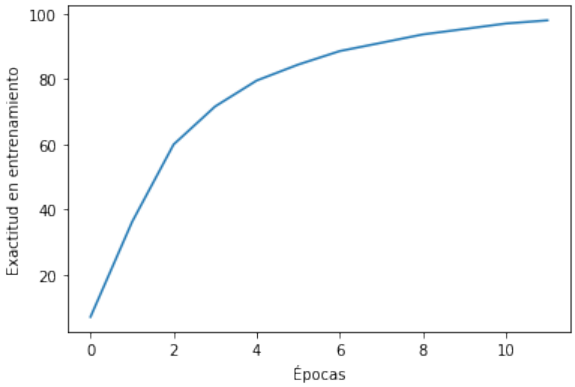

Figura 3: Porcentaje de exactitud de la predicción de la red Neuronal, respecto a la época o iteración, durante el entrenamiento.

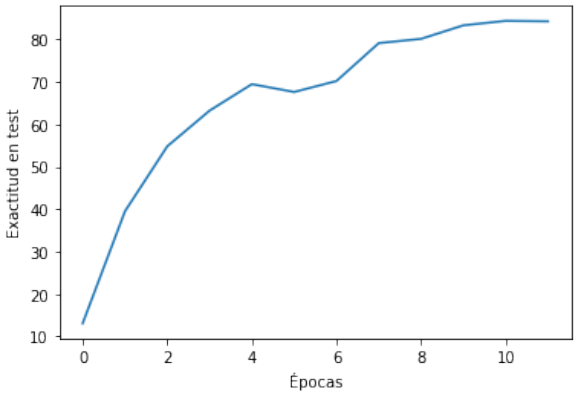

Figura 4: Porcentaje de exactitud de la predicción de la red Neuronal, respecto a la época o iteración, durante la evaluación.

rentable en la exactitud de sus predicciones respecto al costo de tiempo y costo computacional. Aumentar las dimensiones dentro de cada neurona de la red convolucional podría traernos una mayor exactitud en la predicción con un menor coste computacional, debido a que la información que se maneja en paralelo aumentaría, sin embargo los resultados quedarán al pendiente para un futuro proyecto.

Conclusiones

Podemos observar que el porcentaje de exactitud en la predicción de el mensaje que se encuentra dentro de cada audio va aumentando con cada iteración, obteniendo un valor del 97 % durante el entrenamiento y un 84 % durante la evaluación de la red neuronal. Estos resultados nos indican que trabajar con una red convolucional 1D puede darnos buenos resultados con una cantidad razonable de tiempo de entrenamiento.

Agradecimientos

Debo darle agradecimientos a La Asociación de Estudiantes de Pregrado de Ingeniería Física UNI, por la oportunidad de participar en la Escuela de Verano AEPIF, y al BsC. Elvin Muñoz Vega por el apoyo, la dedicación y el compromiso que mostró como tutor durante el curso y la realización de este proyecto.

Referencias

[1] Smith, J.O. 'Example Applications of the DFT - Spectrograms', in *Mathematics of the Discrete Fourier Transform (DFT) with Audio Applications*, Second Edition, http://ccrma.stanford.edu/~jos/mdft/Fourier_Theorems_DFT.html, online book, 2007 edition, accessed <date>.

[2] Poser, W. J. (1990). [Review of *Speech Communication: Human and Machine*, by D. O'Shaughnessy]. *Journal of the International Phonetic Association*, 20(2), 52–54. http://www.jstor.org/stable/44526812

[3] Loncomilla, P. (2016). Deep learning: Redes convolucionales. Recuperado de https://ccc.inaoep.mx/~pgomez/deep/presentations/2016Loncomilla.