UNIVERSIDAD DEL SINÚ
Elías Bechara Zainúm
Seccional Cartagena

# MODELO DE RED NEURONAL CONVOLUCIONAL PARA EL RECONOCIMIENTO DEL ALFABETO EN LENGUAJE DE SEÑAS COLOMBIANO

## MANUAL TECNICO


**PEDRO LUIS TORRES ÁLVAREZ**

**GUSTAVO CASTRO LOZANO**



**UNIVERSIDAD DEL SINÚ ELÍAS BECHARÁ ZAINÚM**

**FACULTAD DE CIENCIAS EXACTAS E INGENIERÍAS**

**ESCUELA INGENIERÍA DE SISTEMAS**

**CARTAGENA, COLOMBIA**

**2019**

## Contenido

## Tabla de ilustraciones

# INTRODUCCIÓN

En el manual que se presenta, usted encontrara todas las herramientas y funcionalidades utilizadas para el desarrollo del modelo de aprendizaje de lenguaje de seña. Es importante leerlo detenidamente antes de diseñar un modelo similar.

El algoritmo consiste en el aprendizaje de un modelo de lenguaje de seña para las personas audio impedida como las que no lo son, el algoritmo está en fase inicial para que en un futuro se pueda seguir mejorando.

# 1. HERRAMIENTAS UTILIZADAS

Para la realización del algoritmo se tuvieron en cuenta las siguientes herramientas.

## 1.1. Google Drive

Servicio de alojamiento de archivos, permite guardar cualquier tipo de archivo y también puede crear archivo. Tiene un almacenamiento limitado de 15 gigabite de almacenamiento gratis.

## 1.2. Google Colaboratory

Esta herramienta es un entorno gratuito de Jupyter Notebook el cual no requiere configuración y no es necesario contar con una maquina física con grandes capacidades, ya que esta herramienta es totalmente ejecutable en la nube. Nos permite escribir, guardar, ejecutar código y hasta compartir el análisis de datos y tienes recursos de información muy potentes todo desde tu navegador y de forma gratuita. Se recomienda usar el navegador de Google Chrome.

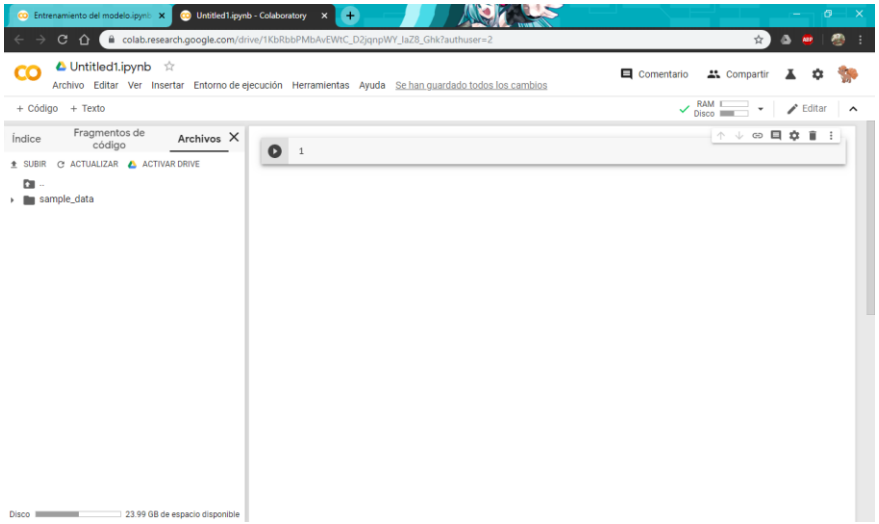


*Ilustración 1 Entorno de Google Colaboratory*

## 2. ALGORITMO

Se muestra como está dividido el algoritmo por cada función detallando el modelo. Muestra desde que se cargan los datos hasta que se guardan el modelo creado en el algoritmo.

### 2.1. Librerías

Teniendo en cuenta el entorno de desarrollo Google Colaboratory, hay que importar todas las librerías necesarias para implementar el algoritmo. Las libres de uso libre que se deben importar se muestran a continuación en la siguiente ilustración (2):

```
[ ]
import os
import keras
import tensorflow
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import optimizers
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import  Convolution2D, MaxPooling2D
from tensorflow.python.keras import backend as K
from keras import layers
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
```

*Ilustración 2 librerías*

### 2.2. Conjunto de datos

En la ilustración 3, Estas líneas de código monta una carpeta temporal de drive a Google Colaboratory. Esta parte se realiza el cargue del conjunto de datos a la herramienta de Google Colaboratory que está guardado en Google Drive. Es necesario para poder entrenar y validar el modelo.

```
[ ]   1 from google.colab import drive
      2 drive.mount('/content/drive')
```

*Ilustración 3 Cargue de conjunto de datos*

En la ilustración 4 se muestra como está distribuido en el Drive el conjunto de imágenes (Abecedario) y el algoritmo (entrenamiento del modelo) que se usó para el modelo.

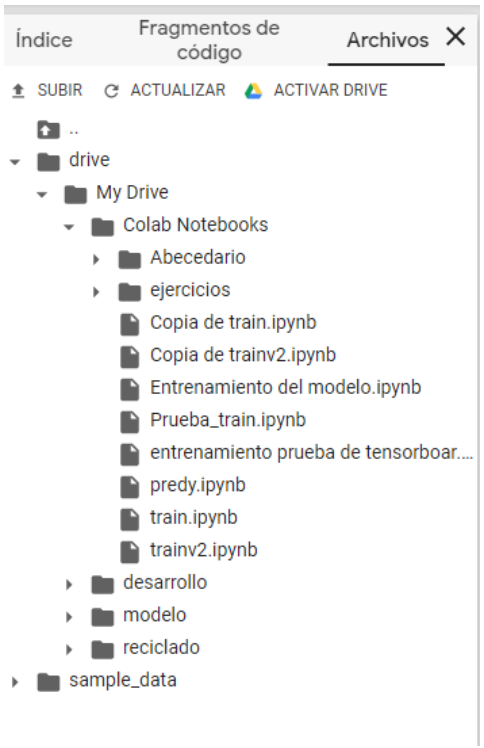


*Ilustración 4 Organización en Drive*

Estos son los parámetros que serán usados en la implementación del algoritmo.

```
"""
Parameters
"""
epocas=15
longitud, altura = 150, 150
batch_size = 32
pasos = 100
validation_steps = 50
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3, 3)
tamano_filtro2 = (2, 2)
tamano_pool = (2, 2)
clases = 26
lr = 0.0004
```

*Ilustración 5 Parámetros*

### 2.3. Preparación de imágenes

En la ilustración (5), se muestra cómo se prepararon el set de datos para usarlo en el entrenamiento del modelo, en la línea 3 de la ilustración es la preparación rescalar la imagen para procesar las imágenes a la escala de los pixeles de datos 1/255, esta normalización permitió que las operaciones matriciales que se realizaron en entrenamiento sean mucho más rápidas y eficientes, el Split de los datos que es un 80% de datos para el entrenamiento de los datos y el 20% de los datos para la validación de la misma.

```
[ ]  1 #Preparamos nuestras imagenes
     2
     3 entrenamiento_datagen = keras.preprocessing.image.ImageDataGenerator(
     4     rescale=1. / 255,
     5     shear_range=0.2,
     6     validation_split=0.2,
     7     horizontal_flip=True)
     8
     9 entrenamiento_generador = entrenamiento_datagen.flow_from_directory(
    10     '/content/drive/My Drive/Colab Notebooks/Abecedario',
    11     target_size=(altura, longitud),
    12     batch_size=batch_size,
    13     class_mode='categorical',
    14     subset='training')
    15
    16 validacion_generador = entrenamiento_datagen.flow_from_directory(
    17    '/content/drive/My Drive/Colab Notebooks/Abecedario',
    18     target_size=(altura, longitud),
    19     batch_size=batch_size,
    20     class_mode='categorical',
    21     subset='validation',
    22      shuffle = False)
```

*Ilustración 6 Preparación de imágenes*

En la línea 9, es el código que se usa para el entrenamiento del modelo muestra el llamado del set de imágenes da como detalle el tamaño de las imágenes que entran y el modo de clase que se usa este caso categorical.

En la línea 16 es el código que se usa para la validación del modelo muestra el llamado del set de imágenes da como detalle el tamaño de las imágenes que entran, el modo de clase que se usa este caso categorical y para que no baraje los datos.

## 2.4. Modelo

```
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1, tamano_filtro1, strides=2, padding ="same", input_shape=(longitud, altura, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Convolution2D(filtrosConv2, tamano_filtro2, strides=2, padding ="same"))
cnn.add(MaxPooling2D(pool_size=tamano_pool))

cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases, activation='softmax'))

cnn.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(lr=lr), metrics=['accuracy'])
```

*Ilustración 7 Modelo*

En la ilustración (6) se definen las funciones que ayudaron a la construcción de la red neuronal convolucional, a continuación, se dará a explicar de forma breve cómo funcionan estos fragmentos de código.

**Sequential ()**

Esta función es una pila de capas lineales, que se crean pasando una lista de instancia de capas al constructor.

**add(convolution2D ())**

Esta capa crea un núcleo de convolución que este entrelazado con la entrada de capas para producir un tensor de salidas. Cuando se usa esta capa como la primera de un modelo, se proporciona ciertas palabras claves cuales son: filters, kernel_zise, strides, padding, input_shape y activation.

- **Filters:** es la dimensión de la salida de la convolucion.
- **Kernel_zise:** especifica las dimensiones de la imagen de entrada.
- **Strides:** zancadas de la convolución a lo largo y ancho.
- **Padding:** "same" proporciona como resultado el relleno de la entrada de modo que la salida tenga la misma longitud que la entrada original.

- **Input_shape:** indica que la entrada esperada serán lotes de vectores de N° de dimensiones.
- **Activation:** las activaciones se pueden usar a través de una capa de activación o puede ser admitido por todas las capas.

**MaxPooling2D**

Operación de agrupación máxima para datos espaciales. MaxPooling2D es un tensor 3D de forma (peso, altura, canal) las dimensiones altura y peso tienden a reducirse a medida que nos adentramos en las capas ocultas de la red neuronal.

- **Pool_size:** tupla de 2 enteros que reduce la escala (vertical, horizontal). (2, 2) reducirá a la mitad la entrada en ambas dimensiones espaciales. Si solo se especifica un número entero, se utilizará la misma longitud de ventana para ambas dimensiones.

**Flatten**

Convierte los elementos de la matriz de imágenes de entrada en un array plano.

**Dense**

Con esta instrucción añadimos una capa oculta (hidden layer) de la red neuronal.

- **Units:** dimensionalidad del espacio de salida.

**Dropout**

La deserción es establecer aleatoriamente una fracción rate de las unidades de entrada en 0 o 1 en cada actualización durante el tiempo de entrenamiento, lo que ayuda a evitar el sobreajuste.

**Compile**

Configura el modelo para el entrenamiento.

- **Optimizer:** instancia del optimizador.
- **Loss:** función de perdida de instancia, si el modelo tiene multiples salidas, puede usar una perdida diferente en cada salida pasando un diccionario o una lista de pérdidas.
- **Metrics:** Lista de métricas que el modelo evaluará durante el entrenamiento y las pruebas.

## 2.5. ENTRENAMIENTO

En la ilustración (7) se muestra el procedimiento para entrenar el modelo en datos generados por lotes, se utiliza el 80% que se guardaron en el método generador, el número de pasos para declarar cuando una epoch o época termina y comienza la otra, la epochs numero de épocas para entrenar el modelo, los datos de validación para evaluar la perdida y las métricas del modelo al final de cada época y validation_steps (pasos de validación) es el número de paso para obtener la validación de los datos antes de detenerse al final de cada época.

```
history = cnn.fit_generator(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,
    validation_steps=validation_steps)
```

*Ilustración 8 Entrenamiento del modelo*

## 2.6. Guardar el modelo

En la ilustración (8), se muestra las líneas de comando para guardar el modelo en Google Colaboratory y pasarlo a Drive.

```
target_dir = './modelo/'
if not os.path.exists(target_dir):
  os.mkdir(target_dir)
cnn.save('./modelo/modelo.h5')
cnn.save_weights('./modelo/pesos.h5')
```

```
mv '/content/modelo' '/content/drive/My Drive'
```

*Ilustración 9 Guardar el modelo*