UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"

INGENIERÍA EN TELECOMUNICACIONES

Fundamentos de programación

Trabajo Opcional:

Redes Neuronales Convolucionales

Clasificación avanzada de imágenes con IA

Integrantes:

- Villegas Toctaguano Mateo Rafael
- Gómez Coro Erick Rafael.

2020-2021

Objetivo

Diseñar un clasificador de imágenes mediante el uso de redes neuronales mediante procesos analíticos y prácticos generalmente utilizando los debidos conceptos informáticos de redes neuronales convolucionales por medio de la aplicación python, y a partir de tensorflow para que finalmente podamos dar soluciones experimentales al ejecutar los algoritmos creados, para aplicarlos en casos reales transmitiendo un solución efectiva.

Introducción

El desarrollo de algoritmos avanzados por medio de redes neuronales, desde el pensamiento analítico computacional hasta conceptuarse frente a herramientas prácticas han tenido un movimiento muy importante en esta era principalmente en la programación orientada a objetos, porque al realizar este tipo de programación solventara de muchas actividades para los distintos desarrollos tecnológicos, puesto que por ello este trabajo permitirá desarrollar las debidas destrezas tecnológicas las cuales nos permiten diseñar algoritmos funcionales directamente en casos básicos y real, para que finalmente puedan dar soluciones con respecto al objetivo propuesto.

Una descripción breve refiriéndonos a la redes neuronales las cuales vamos a emplear en este trabajo son de manera clasificatoria, cuyo objetivo es adecuar cada tipo de dibujo con cada tipo de imagen en este caso números, esto por medio de los pixeles que poseen las imágenes y colocarlas en cada neurona en la programación, partiendonos de un tipo específico de red neuronal utilizada, con sus debidas limitaciones, e instrucciones por medio de herramientas a través de capas ocultas, además de ello emplearemos diversas funciones relevantes para realizar la ejecución respectiva.

4. Metodología utilizada (descripción de librerías de Python, funciones, código completo)

Tensor Flow: es una librería Python para computación numérica rápida creada y publicada por Google. Es una librería que puede utilizarse para crear modelos de Deep Learning directamente o utilizando librerías de envolturas que simplifican el proceso construido sobre Tensor Flow.
mnist.load( ): Carga el conjunto de datos MNIST.
.astype(): El método se usa para convertir un objeto de pandas a un tipo específico de especificado. astype()La función también proporciona la capacidad de convertir cualquier columna existente adecuada a tipo categórico.
matplotlib.pyplot. y ticks (ticks = Ninguno , etiquetas = Ninguno , ** kwargs)
Obtenga o establezca las ubicaciones y etiquetas de marcas actuales del eje y.

No pase argumentos para devolver los valores actuales sin modificarlos.

Parámetros

ticks: en forma de matriz, opcional

La lista de ubicaciones de ytick. Pasar una lista vacía elimina todos los yticks.

etiquetas: en forma de matriz, opcional

Las etiquetas para colocar en las ubicaciones de las marcas dadas . Este argumento solo se puede pasar si también se pasan ticks .

****kwargs:** Las propiedades se pueden utilizar para controlar la apariencia de las etiquetas.
matplotlib.pyplot.grid

nombres_clases=metadatos.features[“label”],names : usa la función .features para etiquetar las 10 posibles categorías que existe en la lista de metadatos descargada previamente.
tf.cast( ): Este método proviene de la librería de tensor flow y convierte un tensor (es un objeto algebraico que describe una relación multilineal entre conjuntos de objetos algebraicos relacionados con un espacio vectorial) en un nuevo tipo.

.map( ): esta función devuelve un objeto de mapa (que es un iterador) de los resultados después de aplicar la función dada a cada elemento de un iterable dado (lista, tupla, etc.).

- sintaxis: mapa (fun, iter)
- Parámetros: **fun :** Es una función a la que map pasa cada elemento de iterable dado.
  ter: es un iterable que se va a mapear.

.caché( ): Es un método que sirve para agregar una serie de datos al cache (usar memoria en lugar de disco, entrenamiento más rápido).
Numpy: Numpy es una librería en la que se define un tipo de dato que representa matrices multidimensionales, equivalentes a las matrices del R. Además incluye algunas funcionalidades básicas para trabajar con ellas.

numpy.take( ): La función numpy.take() devuelve elementos de la matriz a lo largo del eje y los índices mencionados.

.reshape( ): Esta función devuelve un nuevo array con los datos del array cedido como primer argumento y el nuevo tamaño indicado.
np.arg max( ) :Devuelve los índices de los valores máximos a lo largo de un eje.

plt.subplots( ): crea una figura y una cuadrícula de subparcelas con una sola llamada, al tiempo que proporciona un control razonable sobre cómo se crean las parcelas individuales

plt.xticks(): en el módulo pyplot de la biblioteca matplotlib se usa para obtener y establecer las ubicaciones actuales de ticks y etiquetas del eje x.
Sintaxis: matplotlib.pyplot.xticks (ticks = None, etiquetas = None, ** kwargs)

Parámetros: este método acepta los siguientes parámetros que se describen a continuación:

- **ticks:** este parámetro es la lista de ubicaciones de xtick. y un parámetro opcional. Si se pasa una lista vacía como argumento, eliminará todos los xticks
- **etiquetas:** este parámetro contiene etiquetas para colocar en las ubicaciones de ticks dadas. Y es un parámetro opcional.
- **** kwargs:** este parámetro es propiedades de texto que se utiliza para controlar la apariencia de las etiquetas.

plt.imshow( ): función que acepta como primer argumento una variable con estructura de array (o imagen PIL).
matplotlib.pyplot. grilla (visible = none, cuál = 'principal' , eje = 'ambos' , ** kwargs)
show(): Funcion para mostrar el gráfico. Para ello se utiliza la función.

from tensorflow.keras.preprocessing.image import ImageDataGenerator: Genere lotes de datos de imágenes de tensor con aumento de datos en tiempo real.

tf.keras.Sequential( ): agrupa una pila lineal de capas en un archivo
t.f. _ Keras . capas _ Conv2D: Capa de convolución 2D (por ejemplo, convolución espacial sobre imágenes).
tf.keras.layers.Flatten( ): Aplana la entrada. No afecta al tamaño del lote.
t.f. _ Keras . capas _ MaxPool2D: Operación de agrupación máxima para datos espaciales 2D.

Parámetros
booleano visible o ninguno, opcional

Ya sea para mostrar las líneas de cuadrícula. Si se proporcionan kwargs , se supone que desea que la cuadrícula esté activada y visible se establecerá en Verdadero.

Si visible es none no hay kwargs , esto cambia la visibilidad de las líneas.

cuál {'mayor', 'menor', 'ambos'}, opcional

Las líneas de cuadrícula para aplicar los cambios.

eje {'ambos', 'x', 'y'}, opcional

El eje sobre el que aplicar los cambios.

ax.set_xlabel(título) : Añade un título con el contenido de la cadena título al eje x de ax. Se puede personalizar la alineación y la fuente con los mismos parámetros que para el título principal.

tf.keras.layers.Dense( ): Solo su capa NN densamente conectada normal.
split( ): divide un objeto de tipo String en un array (vector) de cadenas mediante la separación de la cadena en subcadenas.

predict( ): nos permite predecir las etiquetas de los valores de los datos sobre la base del modelo entrenado.

```
# Este colab forma parte del video de Redes Neuronales Convolucionales del canal de Youtube "Ringa Tech"
# https://youtu.be/eGDS1W93Bng
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import numpy as np
from tensorflow.keras.utils import to_categorical

#Cargar los datos de MNIST
#Aqui lo hago de otra manera porque es mas simple para poder usar el modulo de aumento de datos
#de Keras de esta manera
(X_entrenamiento, Y_entrenamiento), (X_pruebas, Y_pruebas) = mnist.load_data()
```

```
#Colocar los datos en la forma correcta que ya hemos visto (1, 28, 28, 1)
X_entrenamiento = X_entrenamiento.reshape(X_entrenamiento.shape[0], 28, 28, 1)
X_pruebas = X_pruebas.reshape(X_pruebas.shape[0], 28, 28, 1)

#Hacer 'one-hot encoding' de los resultados (e.g. en lugar de tener como
#...resultado una sola neurona, tendre 10 donde solo el resultado correcto sera 1 y el resto 0)
Y_entrenamiento = to_categorical(Y_entrenamiento)
Y_pruebas = to_categorical(Y_pruebas)

#Convertir a flotante y normalizar para que aprenda mejor la red
X_entrenamiento = X_entrenamiento.astype('float32') / 255
X_pruebas = X_pruebas.astype('float32') / 255
```

```
#Codigo para mostrar imagenes del set, no es necesario ejecutarlo, solo imprime unos numeros :)
import matplotlib.pyplot as plt
filas = 2
columnas = 8
num = filas*columnas
imagenes = X_entrenamiento[0:num]
etiquetas = Y_entrenamiento[0:num]
fig, axes = plt.subplots(filas, columnas, figsize=(1.5*columnas,2*filas))
for i in range(num):
    ax = axes[i//columnas, i%columnas]
    ax.imshow(imagenes[i].reshape(28,28), cmap='gray_r')
    ax.set_title('Label: {}'.format(np.argmax(etiquetas[i])))
plt.tight_layout()
plt.show()
```

```
#Aumento de datos
#Variables para controlar las transformaciones que se haran en el aumento de datos
#utilizando ImageDataGenerator de keras

from tensorflow.keras.preprocessing.image import ImageDataGenerator

rango_rotacion = 30
mov_ancho = 0.25
mov_alto = 0.25
#rango_inclinacion=15 #No uso este de momento pero si quieres puedes probar usandolo!
rango_acercamiento=[0.5,1.5]

datagen = ImageDataGenerator(
    rotation_range = rango_rotacion,
    width_shift_range = mov_ancho,
    height_shift_range = mov_alto,
    zoom_range=rango_acercamiento,
    #shear_range=rango_inclinacion #No uso este de momento pero si quieres puedes probar usandolo!
)
```

```python
#Codigo para mostrar imagenes del set, no es necesario ejecutarlo, solo imprime como se ven antes y despues de las transformaciones
filas = 4
columnas = 8
num = filas*columnas
print('ANTES:\n')
fig1, axes1 = plt.subplots(filas, columnas, figsize=(1.5*columnas,2*filas))
for i in range(num):
    ax = axes1[i//columnas, i%columnas]
    ax.imshow(X_entrenamiento[i].reshape(28,28), cmap='gray_r')
    ax.set_title('Label: {}'.format(np.argmax(Y_entrenamiento[i])))
plt.tight_layout()
plt.show()
print('DESPUES:\n')
fig2, axes2 = plt.subplots(filas, columnas, figsize=(1.5*columnas,2*filas))
for X, Y in datagen.flow(X_entrenamiento,Y_entrenamiento.reshape(Y_entrenamiento.shape[0], 10),batch_size=num,shuffle=False):
    for i in range(0, num):
        ax = axes2[i//columnas, i%columnas]
        ax.imshow(X[i].reshape(28,28), cmap='gray_r')
        ax.set_title('Label: {}'.format(int(np.argmax(Y[i]))))
    break
plt.tight_layout()
plt.show()
```

```python
#Modelo!
modelo = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(10, activation="softmax")
])

#Compilación
modelo.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
```

```python
#Los datos para entrenar saldran del datagen, de manera que sean generados con las transformaciones que indicamos
data_gen_entrenamiento = datagen.flow(X_entrenamiento, Y_entrenamiento, batch_size=32)
```

```python
TAMANO_LOTE = 32

#Entrenar la red. Toma un buen rato! Ve por un café ;)
#Oye suscribete al canal!
print("Entrenando modelo...");
epocas=60
history = modelo.fit(
    data_gen_entrenamiento,
    epochs=epocas,
    batch_size=TAMANO_LOTE,
    validation_data=(X_pruebas, Y_pruebas),
    steps_per_epoch=int(np.ceil(60000 / float(TAMANO_LOTE))),
    validation_steps=int(np.ceil(10000 / float(TAMANO_LOTE)))
)

print("Modelo entrenado!");
```

```
#Exportar el modelo al explorador! (Mas detalle de esto en en mi video de exportacion
modelo.save('numeros_conv_ad_do.h5')

#Convertirlo a tensorflow.js
!pip install tensorflowjs

!mkdir carpeta_salida

!tensorflowjs_converter --input_format keras numeros_conv_ad_do.h5 carpeta_salida
```

5.- Pruebas y resultados (capturas de pantalla)

a) **Cargar los datos de MNIST.**

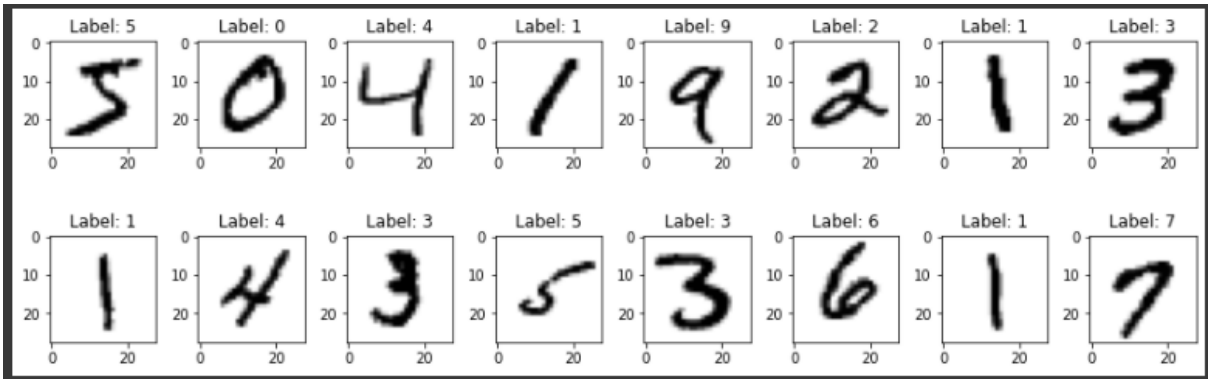
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

b) **Colocar los datos en la forma correcta que ya hemos visto (1, 28, 28, 1)**

c) **Hacer 'one-hot encoding' de los resultados (e.g. en lugar de tener como resultado una sola neurona, tendre 10 donde solo el resultado correcto será 1 y el resto 0**

d) **Convertir a flotante y normalizar para que aprenda mejor la red**

e) **Código para mostrar imágenes del set, no es necesario ejecutarlo, solo imprime unos números.**

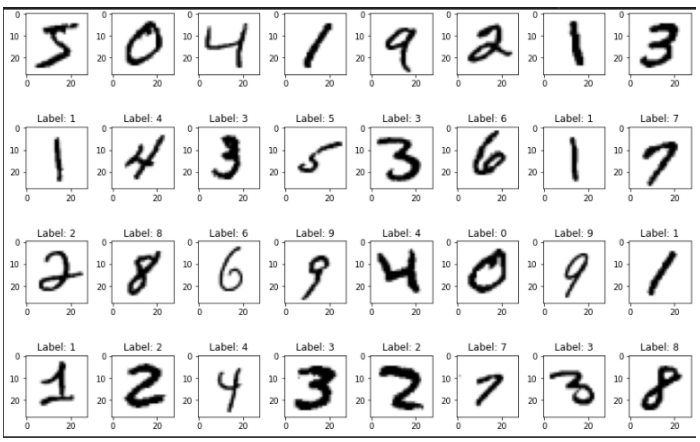


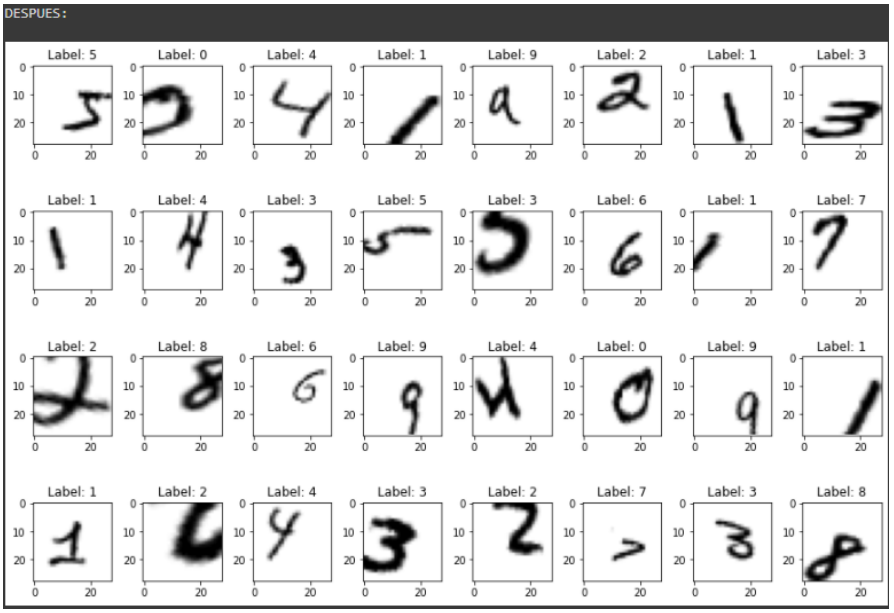
f) **Aumento de datos**

g) **Variables para controlar las transformaciones que se harán en el aumento de datos utilizando ImageDataGenerator de keras**

h) **Código para mostrar imágenes del set, no es necesario ejecutarlo, solo imprime como se ven antes y después de las transformaciones**

Antes:





i) **Modelo**

j) **Compilación**

k) **Los datos para entrenar saldrán del datagen, de manera que sean generados con las transformaciones que indicamos**

l) **Entrenar la red. Toma un buen rato**

```
Entrenando modelo...
Epoch 1/60
1875/1875 [==============================] - 46s 19ms/step - loss: 1.1995 - accuracy: 0.5957 - val_loss: 0.1986 - val_accuracy: 0.9495
Epoch 2/60
1875/1875 [==============================] - 36s 19ms/step - loss: 0.7162 - accuracy: 0.7673 - val_loss: 0.1719 - val_accuracy: 0.9523
Epoch 3/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.5953 - accuracy: 0.8101 - val_loss: 0.1230 - val_accuracy: 0.9621
Epoch 4/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.5369 - accuracy: 0.8249 - val_loss: 0.1055 - val_accuracy: 0.9682
Epoch 5/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.4945 - accuracy: 0.8403 - val_loss: 0.0844 - val_accuracy: 0.9737
Epoch 6/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.4640 - accuracy: 0.8495 - val_loss: 0.0708 - val_accuracy: 0.9773
Epoch 7/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.4474 - accuracy: 0.8550 - val_loss: 0.0748 - val_accuracy: 0.9776
Epoch 8/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.4262 - accuracy: 0.8619 - val_loss: 0.0616 - val_accuracy: 0.9804
Epoch 9/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.4154 - accuracy: 0.8659 - val_loss: 0.0648 - val_accuracy: 0.9788
Epoch 10/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3998 - accuracy: 0.8706 - val_loss: 0.0609 - val_accuracy: 0.9804
Epoch 11/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3934 - accuracy: 0.8723 - val_loss: 0.0638 - val_accuracy: 0.9805
Epoch 12/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3831 - accuracy: 0.8746 - val_loss: 0.0672 - val_accuracy: 0.9790
Epoch 13/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3771 - accuracy: 0.8765 - val_loss: 0.0598 - val_accuracy: 0.9817
Epoch 14/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3677 - accuracy: 0.8819 - val_loss: 0.0510 - val_accuracy: 0.9832
Epoch 15/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.3643 - accuracy: 0.8813 - val_loss: 0.0615 - val_accuracy: 0.9809
Epoch 16/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3545 - accuracy: 0.8856 - val_loss: 0.0579 - val_accuracy: 0.9815
Epoch 17/60
```

```
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3541 - accuracy: 0.8856 - val_loss: 0.0600 - val_accuracy: 0.9820
Epoch 18/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.3428 - accuracy: 0.8901 - val_loss: 0.0806 - val_accuracy: 0.9763
Epoch 19/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3400 - accuracy: 0.8906 - val_loss: 0.0530 - val_accuracy: 0.9835
Epoch 20/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3459 - accuracy: 0.8870 - val_loss: 0.0635 - val_accuracy: 0.9799
Epoch 21/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3383 - accuracy: 0.8907 - val_loss: 0.0484 - val_accuracy: 0.9840
Epoch 22/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3330 - accuracy: 0.8938 - val_loss: 0.0453 - val_accuracy: 0.9859
Epoch 23/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.3281 - accuracy: 0.8942 - val_loss: 0.0633 - val_accuracy: 0.9803
Epoch 24/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3281 - accuracy: 0.8928 - val_loss: 0.0610 - val_accuracy: 0.9811
Epoch 25/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3208 - accuracy: 0.8960 - val_loss: 0.0568 - val_accuracy: 0.9832
Epoch 26/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3173 - accuracy: 0.8977 - val_loss: 0.0704 - val_accuracy: 0.9788
Epoch 27/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3191 - accuracy: 0.8964 - val_loss: 0.0530 - val_accuracy: 0.9842
Epoch 28/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3111 - accuracy: 0.8996 - val_loss: 0.0557 - val_accuracy: 0.9838
Epoch 29/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.3144 - accuracy: 0.8989 - val_loss: 0.0538 - val_accuracy: 0.9838
Epoch 30/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.3083 - accuracy: 0.9001 - val_loss: 0.0484 - val_accuracy: 0.9849
Epoch 31/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.3072 - accuracy: 0.9011 - val_loss: 0.0532 - val_accuracy: 0.9837
Epoch 32/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3097 - accuracy: 0.9003 - val_loss: 0.0533 - val_accuracy: 0.9855
Epoch 33/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.3124 - accuracy: 0.8999 - val_loss: 0.0438 - val_accuracy: 0.9880
Epoch 34/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.3006 - accuracy: 0.9038 - val_loss: 0.0562 - val_accuracy: 0.9835
```

```
Epoch 45/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2883 - accuracy: 0.9071 - val_loss: 0.0441 - val_accuracy: 0.9870
Epoch 46/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2898 - accuracy: 0.9057 - val_loss: 0.0494 - val_accuracy: 0.9847
Epoch 47/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2890 - accuracy: 0.9069 - val_loss: 0.0627 - val_accuracy: 0.9809
Epoch 48/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2834 - accuracy: 0.9090 - val_loss: 0.0672 - val_accuracy: 0.9783
Epoch 49/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2896 - accuracy: 0.9073 - val_loss: 0.0563 - val_accuracy: 0.9809
Epoch 50/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2796 - accuracy: 0.9098 - val_loss: 0.0565 - val_accuracy: 0.9817
Epoch 51/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2832 - accuracy: 0.9074 - val_loss: 0.0716 - val_accuracy: 0.9795
Epoch 52/60
1875/1875 [==============================] - 34s 18ms/step - loss: 0.2814 - accuracy: 0.9100 - val_loss: 0.0703 - val_accuracy: 0.9795
Epoch 53/60
1875/1875 [==============================] - 35s 18ms/step - loss: 0.2841 - accuracy: 0.9083 - val_loss: 0.0578 - val_accuracy: 0.9821
Epoch 54/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2796 - accuracy: 0.9087 - val_loss: 0.0623 - val_accuracy: 0.9813
Epoch 55/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2849 - accuracy: 0.9082 - val_loss: 0.0633 - val_accuracy: 0.9819
Epoch 56/60
1875/1875 [==============================] - 36s 19ms/step - loss: 0.2768 - accuracy: 0.9114 - val_loss: 0.0400 - val_accuracy: 0.9878
Epoch 57/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2827 - accuracy: 0.9087 - val_loss: 0.0530 - val_accuracy: 0.9840
Epoch 58/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2763 - accuracy: 0.9118 - val_loss: 0.0519 - val_accuracy: 0.9843
Epoch 59/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2765 - accuracy: 0.9097 - val_loss: 0.0597 - val_accuracy: 0.9830
Epoch 60/60
1875/1875 [==============================] - 35s 19ms/step - loss: 0.2753 - accuracy: 0.9101 - val_loss: 0.0488 - val_accuracy: 0.9854
Modelo entrenado!
```

m) Exportar el modelo al explorador

n) Convertirlo a tensorflow.js

```
Collecting tensorflowjs
  Downloading tensorflowjs-3.13.0-py3-none-any.whl (77 kB)
     |████████████████████████████████| 77 kB 3.2 MB/s
Requirement already satisfied: tensorflow<3,>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflowjs) (2.8.0)
Requirement already satisfied: tensorflow-hub<0.13,>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from tensorflowjs) (0.12.0)
Requirement already satisfied: six<2,>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflowjs) (1.15.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (0.24.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (57.4.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.1.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (3.3.0)
Requirement already satisfied: tensorboard<2.9,>=2.8 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (2.8.0)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (3.17.3)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (13.0.0)
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (0.5.3)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.21.5)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.13.3)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.1.0)
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (2.8.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (3.10.0.2)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.0.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.6.3)
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (1.44.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow<3,>=2.1.0->tensorflowjs) (0.2.0)
Collecting tf-estimator-nightly==2.8.0.dev2021122109
  Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl (462 kB)
     |████████████████████████████████| 462 kB 11.1 MB/s
```

```
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tensorflow<3,>=2.1.0->tensorflowjs) (0.37.1)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tensorflow<3,>=2.1.0->tensorflowjs) (1.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (2.23.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (1.8.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (0.6.1
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (1.0.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (0.4.6)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (1.35.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (3.3.6)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (4.8)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0-
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (4.
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensor
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow<3,
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (3.0.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow<3,>=2.1.0->tensorflowjs) (20
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tens
Installing collected packages: tf-estimator-nightly, tensorflowjs
Successfully installed tensorflowjs-3.13.0 tf-estimator-nightly-2.8.0.dev2021122109
```

Limpiar Predecir

Modo 1 (Denso)	Modo 2 (Conv)	Modo 3 (Conv+AD+DO)	Modo 4 (Conv+AD+DO) 200
4	4	9	9

Limpiar Predecir

Modo 1 (Denso)	Modo 2 (Conv)	Modo 3 (Conv+AD+DO)	Modo 4 (Conv+AD+DO) 200
8	7	7	7

6.- Conclusiones y recomendaciones:

- El modo que se trabaja es por medio de redes neuronales convolucionales a través de imágenes basándonos en la admisión de los píxeles de la imagen y a su vez extrayendo las características de la misma para clasificar de manera directa a través de regulares que se crearán posteriormente.

- El tipo de red neuronal que se hizo énfasis para el desarrollo del código principalmente es la red neuronal convolucional la cual vamos a manipular sobre las redes neuronales regulares, herramientas para que pueda resolver problemas más complejos y no esté limitada a sólo resolver ecuaciones lineales, esto por medio de la estimulación y manipulación de las capas ocultas y la integración de capas una de convolución y otra capa de agrupación.

- Las capas tanto de convolución y otra agrupación son las mismas que estimulan a las redes neuronales realizar la extracción de las características de la imagen para que posteriormente se realice una clasificación mediante las capas regulares.

- Las redes neuronales descomponen las entradas en capas de abstracción. Se pueden entrenar con muchos ejemplos para que reconozcan patrones de voz o en imágenes, Su comportamiento está definido por la forma en que se conectan sus elementos individuales, así como por la importancia de dichas conexiones.

- Una importante recomendación que hace énfasis en la manipulación de redes neuronales a través de problemas con un nivel de dificultad es simplemente que realicemos las operaciones por medio y a través de diversas capas ocultas, generalmente entre más alto sea el grado de dificultad para resolver los problemas mas alta sera el numero de capas ocultas que se manipulara y menos y se tiene un grado menor.

7.- Referencias de consulta:

- https://unipython.com/introduccion-a-tensorflow/#:~:text=TensorFlow%20es%20una%20librer%C3%ADa%20Python,el%20proceso%20construido%20sobre%20TensorFlow.
- https://www.tensorflow.org/api_docs/python/tf/cast
- https://www.geeksforgeeks.org/python-map-function/
- https://alvarohurtado.es/2013/06/09/funciones-en-python-con-cache/
- https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html
- https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D