Clasificación de Dígitos

UPC

2021-1

Presentado Por:

- Cesar Mosqueira - u201910750
- Paola Viviana Angeles Lliuya - u20191A919
- Angel Gustavo Chuco - u201613694

1. Descripción del problema

El problema elegido fue la clasificación de dígitos escritos a mano. Es un problema introductorio a computer vision, una rama muy importante de la inteligencia artificial. Con ella se pueden computarizar una infinidad de opciones y se puede entrenar a computadoras para que identifiquen informacion en una imagen. La clasificacion de dígitos escritos a mano es un pequeño uso que se le puede dar.

En la actualidad hay una gran cantidad de datos disponibles y cada vez va más en aumento en la era del internet. Hay tareas de clasificación con un gran número de clases como es el caso del reconocimiento de caracteres manuscritos chinos con más de 6 mil clases, surge un problema de categorización de documentos en la web cuando son procesados gigabytes de datos con alta dimensión. [1].

El problema de la clasificación es el más frecuente en la práctica, de esta forma, construiremos un modelo que permite predecir la categoría de las instancias en función de una serie de atributos de entrada. La clase se convertirá en variable objetivo a predecir. [2].

2. Dataset

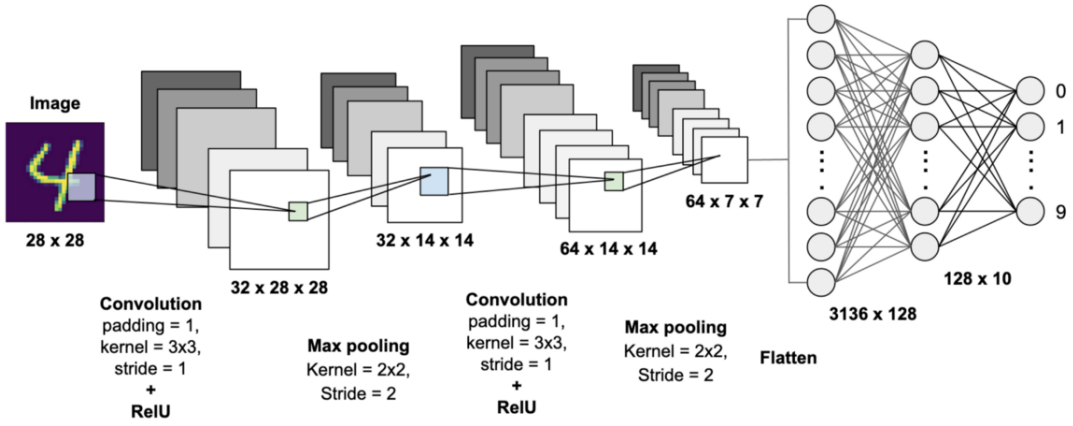
El dataset usado es uno muy conocido. MNIST[1], es una base de datos que contiene 70000 imágenes para entrenar redes dedicadas a computer vision a reconocer dígitos escritos a mano. 60000 imágenes son dedicadas al entrenamiento y otras 10000 al test de la red neuronal.

A continuación detallamos nuestro dataset en la siguiente imagen:

[1]'Modified National Institute of Standards Technology'. La version normalizada a 28x28 y con antialiasing de la antigua base de datos de NIST

3. El modelo

La red neuronal planteada contiene 4 capas. El input que recibe es un tensor que representa la imagen en grayscale (28x28) y el output es un tensor de 10 componentes donde cada uno es la probabilidad de que la imagen se trate del numero correspondiente a su óndice.



La descripción de cada una de las capas es la siguiente:

Capa de NN	Función (uso)
Dense	Capa 'oculta' regular
Conv2D	Se encarga de mover el kernel por la imagen y generar un feature map
MaxPooling2D	Se encarga de encoger el input para hacer el output mas robusto
Dropout	Convierte inputs random a 0 para evitar overfitting
Flatten	Convierte las N dimensiones del input en una sola

4. Aplicación

El modelo se desarrollo en Tensorflow y se hace uso del lenguaje Python. TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. [4].

Para el entrenamiento es necesario 2 tensores. El que contenga las imagenes de 28x28 y uno del mismo tamaño y dimension que contenga los valores que representan las imágenes. Es necesario que el arreglo de valores este categorizado[2], para que se pueda comparar con el output de la red neuronal. Las imágenes deben ser de 28x28 y deben tener un solo componente de color. La función de los que usamos es categorical cross entropy y el optimizador es ADAM, debido a su capacidad de manejar mejor las imágenes con ruido a diferencia de SGD[3] ya que se adapta al momento en el que se esta analizando el gradiente [3].

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 26, 26, 32)
conv2d$_1$($Conv2D$)	(None, 24, 24, 64)
max$_p$$ooling2d$($MaxPooling2D$)	(None, 12, 12, 64)
dropout (Dropout)	(None, 12, 12, 64)
flatten (Flatten)	(None, 9216)
dense (Dense)	(None, 128)
dropout$_1$($Dropout$)	(None, 128)
dense$_1$($Dense$)	(None, 10)

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
     activation='relu',
     input_shape=(28, 28, 1)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

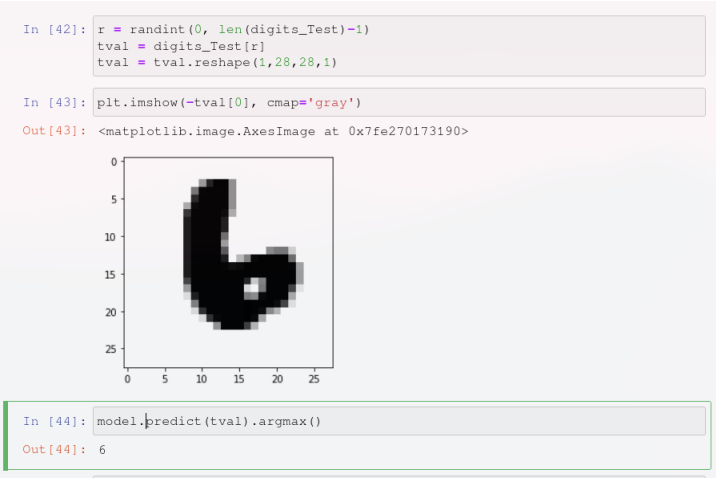
Con esta configuración y, en 10 epochs, se logró un accuracy de 99.15 %. La información del entrenamiento está disponible en el notebook del repositorio. El modelo entrenado se guardo en un archivo .h5 en el mismo repo. Archivo el cual puede ser cargado y utilizado con:

```
from tensorflow.keras.models import load_model
model = load_model('/content/drive/MyDrive/DigitRecognizer/DigitRecognizer.h5'
model.compile(loss='categorical_crossentropy',
      optimizer='adam',
      metrics=['accuracy'])
```

Y estará listo para usar

```
model.predict(<np.ndimarray shape=(n,28,28,1)>)
```

[2]Por ejemplo si el valor es 3, debera representarse como [0,0,0,1,0,0,0,0,0,0]
[3]Stochastic Gradient Descent



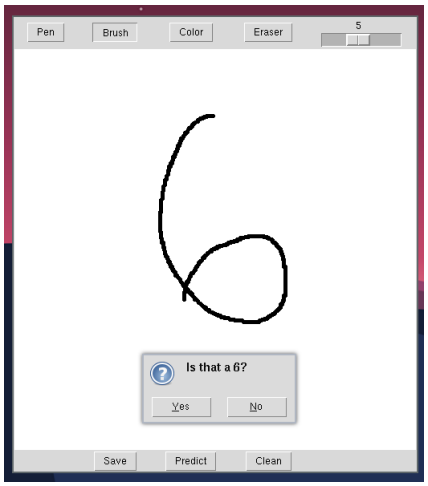


Figura 1: Pruebas en Notebook y GUI

Funcion la cual retornará un tensor de 10 elementos donde el componente de mayor valor será el número que predijo la red.

5. Uso y Ejecución

Definitivamente este modelo de entrenamiento de redes convolucionales puede ser usado en una infinidad de casos. Pero es muy importante un dataset para realizar tal tarea. Hicimos una pequeña aplicacion de prueba usando solamente Tkinter para el GUI y Pillow para procesar la imagen que ingrese el usuario y guardarla.

La aplicación GUI también necesitará de tensorflow solamente para cargar el archivo .h5 y hacer la predicción. Para cargar el modelo en la aplicación realizamos lo siguiente:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras


from PIL import Image, ImageDraw, ImageOps, ImageFilter


model = keras.models.load_model('DigitRecognizer.h5')
model.compile(loss='categorical_crossentropy',
      optimizer='adam',
      metrics=['accuracy'])

def Predict(path):
    im = Image.open(path)
    im = ImageOps.grayscale(im.filter(ImageFilter.CONTOUR))
    im = im.resize((28,28))
    im = np.asarray(im)
    im = im/255
    im = im.reshape(1,28,28,1)
    return model.predict(im).argmax()
```

6. Capturas de prueba

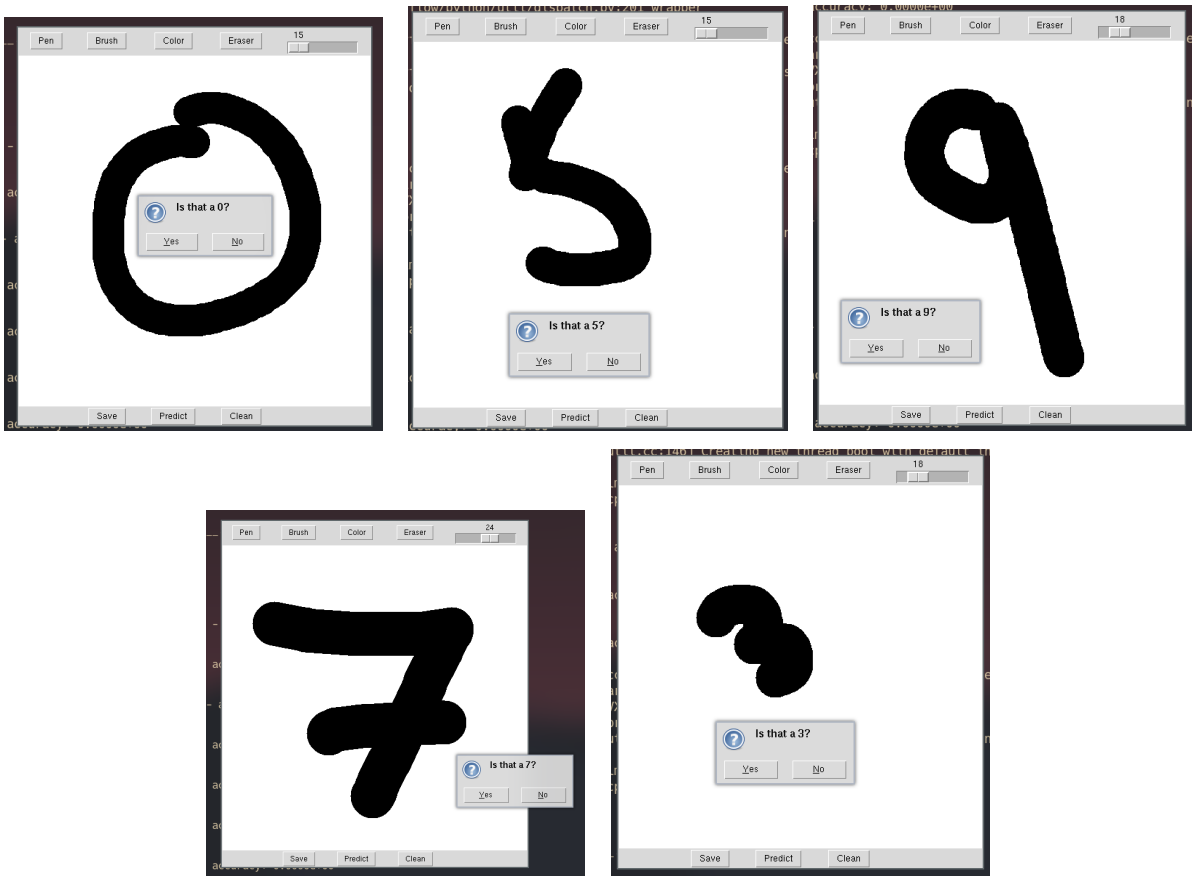


Figura 2: Pruebas del programa en ejecucion.

Referencias

[1] Euler Tito Chura Carlos Alberto Silva Delgado. Clasificación de dígitos manuscritos de imágenes digitales, Mar 2015.

[2] M. Nielsen. Neural networks and deep learning, Jun 2015.

[3] Tony Peng SyncedReview. Fast as adam good as sgd' — new optimizer has both, May 2019.

[4] Wikipedia. Tensorflow, artificial intelligence wiki, Jun 2021.