

Clasificacion de Digitos

UPC

2021-1

Presentado Por:

- Cesar Mosqueira
- Viviana Angeles Lliuya
- Angel Gustavo Chuco

1. Descripcion del problema

El problema elegido fue la clasificacion de digitos 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 digitos escritos a mano es un pequeño uso que se le puede dar.

2. Dataset

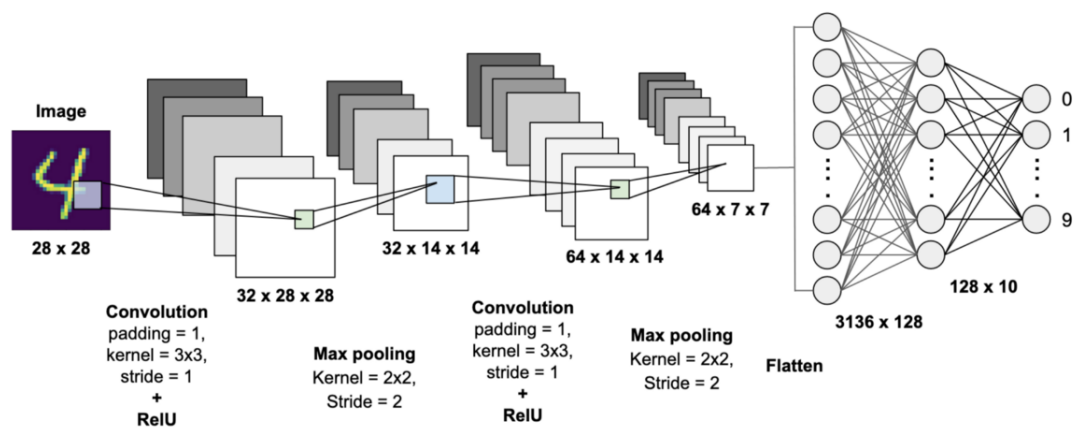
El dataset usado es uno muy conocido. MNIST¹, es una base de datos que contiene 70000 imagenes para entrenar redes dedicadas a computer vision a reconocer digitos escritos a mano. 60000 imagenes son dedicadas al entrenamiento y otras 10000 al test de la red neuronal.



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 indice.



La descripcion 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. Aplicacion

El modelo se desarrollo en Tensorflow. 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 imagenes. Es necesario que el arreglo de valores este categorizado², para que se pueda comparar con el output de la red neuronal. Las imagenes deben ser de 28x28 y deben tener un solo componente de color. La funcion de loss que usamos es categorical cross entropy y el optimizador es ADAM, debido a su capacidad de manejar mejor las imagenes con ruido a diferencia de SGD³ ya que se adapta al momento en el que se esta analizando el gradiente [1].

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

³Stochastic Gradient Descent

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 26, 26, 32)
conv2d ₁ (Conv2D)	(None, 24, 24, 64)
max _{pooling} 2d (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)

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
3
4 model = Sequential()
5 model.add(Conv2D(32, kernel_size=(3, 3),
6     activation='relu',
7     input_shape=(28, 28, 1)))
8
9 model.add(Conv2D(64, (3, 3), activation='relu'))
10 model.add(MaxPooling2D(pool_size=(2, 2)))
11 model.add(Dropout(0.25))
12 model.add(Flatten())
13 model.add(Dense(128, activation='relu'))
14 model.add(Dropout(0.5))
15 model.add(Dense(10, activation='softmax'))
```

Con esta configuracion y, en 10 epochs, se logro un accuracy de 99.15%. La informacion del entrenamiento esta 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:

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

Y estara listo para usar

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

Funcion la cual retornara un tensor de 10 elementos donde el componente de mayor valor sera el numero que predijo la red.

5. Uso y Ejecucion

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 aplicacion GUI tambien necesitara de tensorflow solamente para cargar el archivo .h5 y hacer la prediccion.

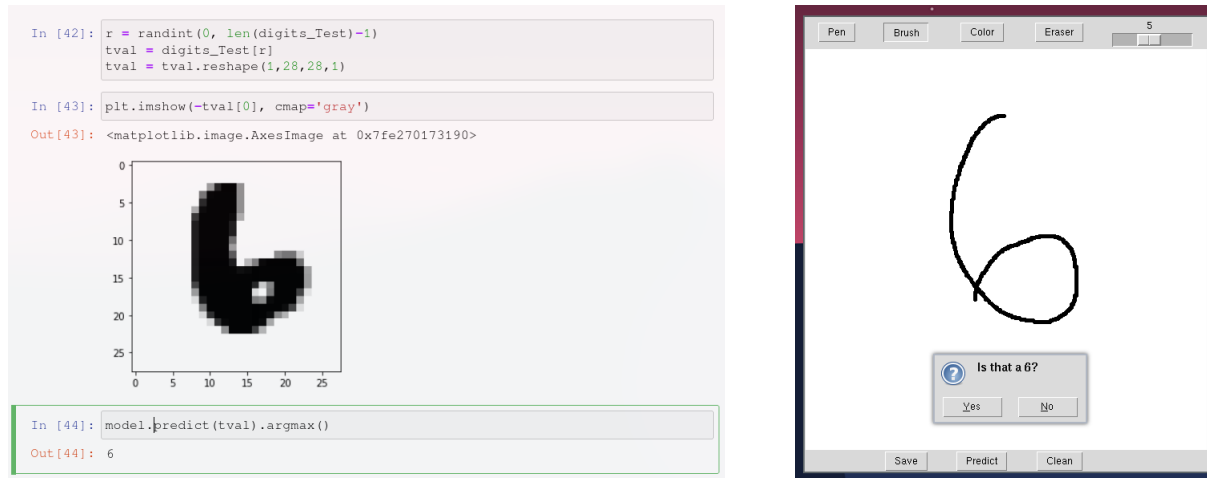


Figura 1: Pruebas en Notebook y GUI

Para cargar el modelo en la aplicación GUI:

```
1 import numpy as np
2 import tensorflow as tf
3 from tensorflow import keras
4
5
6 from PIL import Image, ImageDraw, ImageOps, ImageFilter
7
8
9 model = keras.models.load_model('DigitRecognizer.h5')
10 model.compile(loss='categorical_crossentropy',
11               optimizer='adam',
12               metrics=['accuracy'])
13
14 def Predict(path):
15     im = Image.open(path)
16     im = ImageOps.grayscale(im.filter(ImageFilter.CONTOUR))
17     im = im.resize((28,28))
18     im = np.asarray(im)
19     im = im/255
20     im = im.reshape(1,28,28,1)
21     return model.predict(im).argmax()
```

Referencias

- [1] Tony Peng SyncedReview. Fast as adam good as sgd' — new optimizer has both, Mar 2019.