Reporte_CNN_Abecedario

June 13, 2022

1 Ejercicio Propuesto CNN Abecedario

Para el ejercicio se pidio crear un "Data Set Propio" con imagenes de distintos abecedarios. Por ello realice una selección de 4 abecedarios los cuales nos serviran de entrada para el entrenamiento de la CNN.

Se obtuvieron entre 5,000 y 9,000 imagenes de cada letra/número, a partir de videos de planas escritas o letras de molde generadas por computadora, extrayendo las letras individualmente. Las imágenes tienen un tamaño de 50x50 px, con un total de fotos de 418,304.

Por lo tanto podemos concluir que el objetivo es que nuestra "Red Neuronal Convolucional" aprenda a clasificar, por sí misma, de qué tipo de letra/número se trata, dando una imagen con la cual no se entreno o testeo en la creación de nuestra red.

2 Generar Data Set

Para generar mi Data Set me apoye de planas escritas y abecedarios generados por computadora con letras de molde, para después por medio de filtros limpiar la imágen y disminuir ruido, por ultimo aplicar una redimensión al 50x50.

Giramos la imagen desde -45 grados hasta 45 grados y translade sobre el eje horizontal (x), unicamente fue en este eje para no perder características escenciales (especpificas) de las letras/números, dichos movimientos nos aumentaron en gran escala nuestro número de imégenes dentro de cada clase de nuestro DataSet

2.1 Importar Librerías

[ ]:
```
import cv2 as cv
import math
import numpy as np
```

2.2 Función Imágenes

Nos permite girar, trasladar y guardar las imágenes generadas.

Cada linea de código contiene el comentario de lo que realiza, así como los parametros que se tiene que usar en cada función en caso de que fueran confusos, decidi agregarlo para mayor comprensión del código.

```
[ ]: def Imagenes(Img, General, ejex, ejey):

    #ESTABLECEMOS RUTA DE IMAGEN Y SE LEVANTA
    img1 = cv.imread('C:\\Users\\gilba\\Desktop\\IA\\DataSet\\'+str(Img))

    #ELIMINAMOS RUIDO DE LA IMAGEN CON FUNCION GAUSSIANBLUR
    #PRIMER PARAMETRO IMAGEN A LIMPIAR
    #SEGUNDO PARAMETRO TAMANIO DE KERNEL, EN ESTE CASO 5 X 5 PX SOLO QUEREMOS␣
    ↪ELIMIANR RUIDO NO QUEREMOS VERLA BORROSA
    #TERCER PARAMETEO ESPECIFICA LOS LIMITES DE LA IMAGEN
    gaussiana = cv.GaussianBlur(img1, (5,5), 0)

    #CONVERTIMOS DE RGB A GRISES
    #PRIMER PARAMETRO IMAGEN A CONVERTIR
    #SEGUNDO PARAMETRO DE QUE ESCALA A QUE ESCALA
    imageOut = cv.cvtColor(gaussiana, cv.COLOR_BGR2GRAY)

    ancho = img1.shape[0]
    alto = img1.shape[1]

    ##APLICAMOS FILTRO BLANCO Y NEGRO PARA BINARIZAR LA LETRA Y ELIMINAR RUIDO
    #MIDE FILAS Y COLUMNAS DE LA IMAGEN
    for x in range(imageOut.shape [0]):
        for y in range (imageOut.shape[1]):
            if imageOut[x,y] < 150:
                imageOut[x,y] =  255 #PONE EL PIXEL EN BLANCO
            else:
                imageOut[x,y] = 0 #SINO LO PONE NEGRO

    #GENERAMOS LA MATRIZ DE TRANSFORMACION DE LA IMAGEN, PARA TRANSLADARLA
    M= np.float32([[1,0,ejex], [0,1,ejey]])

    #WARPAFFINE TRANSLADO POR MEDIO DE OPERACIONES MATRICIALES
    #PRIMER PARAMETRO LA IMAGEN A MODIFICAR
    #SEGUNDO PARAMATRO MATRIZ TRANSFORMADA
    #TERCER PARAMETRO ANCHO Y ALTO DE LA IMGEN
    img = cv.warpAffine(imageOut, M, (ancho, alto))

    #DATOS DE IMAGEN
    #SACAMOS ALTURA Y ANCHURA DE LA IMAGEN HEIGTH, WEIGHT
    h,w = img.shape[:2]
    #FUNCION NUMPY ZEROS SIRVE PARA CREAR UNA MATRIZ DE CEROS
    imgz = np.zeros((h*2, w*2), dtype = 'uint8')

    #CONTADOR DE GUARDADO Y GRADOS DE INCLINACION DE IMAGEN
    num_img = -45
```

```
    ## GIRAMOS LA IMAGEN
    ## GIRA LA LETRA - GIRO A MANECILLAS DEL RELOJ
    ##              + GIRO EN CONTRA DE MANECILLAS
    while(num_img != 41):

        #GIRAMOS LA IMAGEN CON LA FUNCION GETROTATION
        #PRIMER PARAMETRO NOS SIRVE PARA CENTRAR LA IMAGEN
        #SEGUNDO PARAMETRO GRADO DE GIRO
        #TERCER PARAMETRO FACTOR ESCALA
        mw = cv.getRotationMatrix2D( (h//2, w//2), num_img, 1 )

        #DEFINIMOS IMAGEN DE SALIDA, AFINAMOS CON WARPAFFINE
        #PRIMER PARAMETRO IMAGEN A MODIFICAR
        #SEGUNDO PARAMETRO LA MATRIZ TRANSFORMADA
        #TERCER PARAMETRO TAMANIO DE LA IMAGEN DE SALIDA
        imgz = cv.warpAffine(img,mw,(h,w))

        numero = num_img +950
        ruta = 'C:
↪\\Users\\gilba\\Desktop\\IA\\DataSet\\'+str(General)+str(ejex)+'_'+str(ejey)+'_'+␣
↪str(numero)+').jpg'
       cv.imwrite(ruta, imgz)
       num_img += 1
```

2.3 Automatización del Proceso

Como en el código anterior al esté ser un poco extenso y confuso decidi comentarlo sobre el mismo código para que nos sirve cada función, así como los parametros que le envian a cada una. Esto para favorecer la comprensión.

[ ]:
```
#ESTABLECEMOS LA RUTA GENERAL DE LA CARPETA
General = [
    "0\\0 ","1\\1 ","2\\2 ","3\\3 ","4\\4 ","5\\5 ","6\\6 ","7\\7␣
↪","8\\8 ","9\\9 ","AMayus\\A ","aMinus\\a ",
    "BMayus\\B ","bMinus\\b ","CMayus\\C ","cMinus\\c ","DMayus\\D␣
↪","dMinus\\d ","EMayus\\E ","eMinus\\e ","FMayus\\F ","fMinus\\f␣
↪","GMayus\\G ","gMinus\\g ",
    "HMayus\\H ","hMinus\\h ","IMayus\\I ","iMinus\\i ","JMayus\\J␣
↪","jMinus\\j ","KMayus\\K ","kMinus\\k ","LMayus\\L ","lMinus\\l␣
↪","MMayus\\M ","mMinus\\m ",
    "NMayus\\N ","nMinus\\n ","NNMayus\\NN ","nnMinus\\nn ","OMayus\\O␣
↪","oMinus\\o ","PMayus\\P ","pMinus\\p ","QMayus\\Q ","qMinus\\q␣
↪","RMayus\\R ","rMinus\\r ",
    "SMayus\\S ","sMinus\\s ","TMayus\\T ","tMinus\\t ","UMayus\\U␣
↪","uMinus\\u ","VMayus\\V ","vMinus\\v ","WMayus\\W ","wMinus\\w␣
↪","XMayus\\X ","xMinus\\x ",
```

```
    "YMayus\\Y (","yMinus\\y (","ZMayus\\Z (","zMinus\\z ("
]

#ESTABLECEMOS LA RUTA GENERAL DE LA IMAGEN
Img = [
    "03.jpg","13.jpg","23.jpg","33.jpg","43.jpg","53.jpg","63.jpg","73.jpg","83.
 ↪jpg","93.jpg","A3.jpg","am3.jpg",
    "B3.jpg","bm3.jpg","C3.jpg","cm3.jpg","D3.jpg","dm3.jpg","E3.jpg","em3.
 ↪jpg","F3.jpg","fm3.jpg","G3.jpg","gm3.jpg",
    "H3.jpg","hm3.jpg","I3.jpg","im3.jpg","J3.jpg","jm3.jpg","K3.jpg","km3.
 ↪jpg","L3.jpg","lm3.jpg","M3.jpg","mm3.jpg",
    "N3.jpg","nm3.jpg","NN3.jpg","nnm3.jpg","O3.jpg","om3.jpg","P3.jpg","pm3.
 ↪jpg","Q3.jpg","qm3.jpg","R3.jpg","rm3.jpg",
    "S3.jpg","sm3.jpg","T3.jpg","tm3.jpg","U3.jpg","um3.jpg","V3.jpg","vm3.
 ↪jpg","W3.jpg","wm3.jpg","X3.jpg","xm3.jpg",
    "Y3.jpg","ym3.jpg","Z3.jpg","zm3.jpg"
]

#MATRIZ DE TRANSLACION DE IMAGEN DESDE (-1,-1) HASTA (1,1)
lados = [[1,-1,1,-1],[1,1,-1,-1]]

#INICIALIZACION UNICAMENTE PARA PODER CONTINUAR
x = 1
y = 1

#CICLO, RECORREMOS EL ARREGLO DE RUTA GENERAL Y RUTA DE LA IMAGEN PARA LOGRAR␣
 ↪LA AUTOMATIZACION DEL PROCESO
size = len(General)
for h in range(size):
    for i in range(4):
        for j in range(10):

            #CALCULAMOS LA TRANSLACION QUE LE DARA A LA IMAGEN POR CICLO
            #SE MULTIPLICA POR J PARA DARLE UNA TRANSLACION DE HASTA 10 PIXELES␣
 ↪DE IZQUIERDA A DERECHA, ARRIBA A ABAJO
            #PRIMERO EN LA TRANSLACION EJE X
            x = lados [0][i] * (j)

            #SEGUNDO TRANSLACION EJE Y
            #y = lados [1][i] * (j) SOLO USE TRANSLACION EJE X, YA QUE SI USABA␣
 ↪LA TRANSLACION EJE Y SE PERDIAN
            #CARACTERISTICAS ESPECIFICAS
            #DE ALGUNAS IMAGENES COMO LA 'NN'
            Imagenes(Img[h], General[h],x,y)

##FINALIZAR
```

```
print ("PROCESO FINALIZADO")
```

Dicho código nos genera un aproximado de 1,800 imagenes de salida con una sola imagen de entrada, puede variar dependiendo el ángulo de giro que desees darle, así como la translación.

De está manera se logro constituir el Data Set para que sirviera de entrada a nuestro Convolutional Neural Networks, la cual se explica como se llevo acabo en el siguiente apartado.

3 Convolutional Neural Networks Abecedario

Crearemos una Convolutional Neural Networks con Keras y Tensorflow para reconocer imagenes de distintas letras y números.

3.1 Importar Librerías

Importación de librerias necesarias para realizar el ejercicio

[1]:
```
import cv2 as cv
import numpy as np
import os
import re
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

[2]:
```
import keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.python.keras.models import Sequential,Input,Model
#from keras.layers import Dense, Dropout, Flatten
#rom keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten,␣
 ↪Dropout, Dense, Conv2D
)
from tensorflow.python.keras.layers.advanced_activations import LeakyReLU
```

3.2 Cargar set de Imágenes

El proceso 'plt.imread(filepath)' cargará a memoria en un array las 418,304 imágenes, esté proceso puede llevar varios minutos, depende el procesamiento y la memoria RAM de tu computador.

Al cargar las imágenes si no tienen un buen tratamiento previo, puede que te de errores de canales de color más adelante, esto es subjetivo dependiendo los canales de color que contengan tus imagenes, en mi caso, mis imágenes contenian dos canales de color (GRAY), al levantarlas no me generaba error alguno, hasta la hora del entrenamiento de la neurona generaba un error de canales, el cual

solucione con la linea “image = cv.cvtColor(image, cv.COLOR_GRAY2RGB)”, que convertía la imágen de 2 canales a 3 tres canales de color.

[3]:
```
dirname = os.path.join(os.getcwd(),'\\Users\\gilba\\Desktop\\IA\\DataSet')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|png|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            image = plt.imread(filepath)
            #VALIDACIÓN DE 3 CANALES, AGUNAS LAS LEVANTA EN 2, (NULL, 50, 50)
            image = cv.cvtColor(image, cv.COLOR_GRAY2RGB)
            images.append(image)
            b = "Leyendo..." + str(cant)
            print (b, end="\r")
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
                dircount.append(cant)
                cant=0
dircount.append(cant)

dircount = dircount[1:]
dircount[0]=dircount[0]+1
print('Directorios leidos:',len(directories))
print("Imagenes en cada directorio", dircount)
print('suma Total de imagenes en subdirs:',sum(dircount))
```

```
leyendo imagenes de  C:\Users\gilba\Desktop\IA\DataSet\
C:\Users\gilba\Desktop\IA\DataSet\0 1
C:\Users\gilba\Desktop\IA\DataSet\1 6536
C:\Users\gilba\Desktop\IA\DataSet\2 6536
C:\Users\gilba\Desktop\IA\DataSet\3 6536
C:\Users\gilba\Desktop\IA\DataSet\4 6536
C:\Users\gilba\Desktop\IA\DataSet\5 6536
C:\Users\gilba\Desktop\IA\DataSet\6 6536
C:\Users\gilba\Desktop\IA\DataSet\7 6536
```

```
C:\Users\gilba\Desktop\IA\DataSet\8 6536
C:\Users\gilba\Desktop\IA\DataSet\9 6536
C:\Users\gilba\Desktop\IA\DataSet\AMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\aMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\BMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\bMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\CMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\cMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\DMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\dMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\EMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\eMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\FMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\fMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\GMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\gMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\HMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\hMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\IMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\iMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\JMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\jMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\KMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\kMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\LMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\lMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\MMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\mMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\NMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\nMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\NNMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\nnMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\OMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\oMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\PMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\pMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\QMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\qMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\RMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\rMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\SMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\sMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\TMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\tMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\UMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\uMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\VMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\vMinus 6536
```

```
C:\Users\gilba\Desktop\IA\DataSet\WMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\wMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\XMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\xMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\YMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\yMinus 6536
C:\Users\gilba\Desktop\IA\DataSet\ZMayus 6536
C:\Users\gilba\Desktop\IA\DataSet\zMinus 6536
Directorios leidos: 64
Imagenes en cada directorio [6537, 6536, 6536, 6536, 6536, 6536, 6536, 6536,
6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536,
6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536,
6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536,
6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536, 6536,
6536, 6536, 6536, 6535]
suma Total de imagenes en subdirs: 418304
```

3.3 Creamos las etiquetas

Las etiquetas se crean en labels, es decir cada letra tendra un valor entre 0 y 63. Esto se realiza para poder hacer uso de el algoritmo supervisado e indicar que cuando cargamos una imagen de Dhalia ya sabemos que corresponde a la etiqueta 0. Con estos parametros de entrada y salida, la red tendrá la capacidad de realizar el ajuste en los pesos de las neuronas.

[4]:
```
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
print("Cantidad etiquetas creadas: ",len(labels))
```

```
Cantidad etiquetas creadas:  418304
```

[5]:
```
letras=[]
indice=0
for directorio in directories:
    name = directorio.split(os.sep)
    print(indice , name[len(name)-1])
    letras.append(name[len(name)-1])
    indice=indice+1
```

```
0 0
1 1
2 2
3 3
4 4
5 5
6 6
```

```
7 7
8 8
9 9
10 AMayus
11 aMinus
12 BMayus
13 bMinus
14 CMayus
15 cMinus
16 DMayus
17 dMinus
18 EMayus
19 eMinus
20 FMayus
21 fMinus
22 GMayus
23 gMinus
24 HMayus
25 hMinus
26 IMayus
27 iMinus
28 JMayus
29 jMinus
30 KMayus
31 kMinus
32 LMayus
33 lMinus
34 MMayus
35 mMinus
36 NMayus
37 nMinus
38 NNMayus
39 nnMinus
40 OMayus
41 oMinus
42 PMayus
43 pMinus
44 QMayus
45 qMinus
46 RMayus
47 rMinus
48 SMayus
49 sMinus
50 TMayus
51 tMinus
52 UMayus
53 uMinus
54 VMayus
```

```
55 vMinus
56 WMayus
57 wMinus
58 XMayus
59 xMinus
60 YMayus
61 yMinus
62 ZMayus
63 zMinus
```

Las etiqueas se convierten en Array con numpy.array

[6]:
```
y = np.array(labels)
X = np.array(images, dtype=np.uint8) #convierto de lista a numpy


# Find the unique numbers from the train labels
classes = np.unique(y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Total number of outputs :  64
Output classes :  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63]
```

3.4 Creamos Sets de Entrenamiento y Test

Debemos destacar que en la forma de los array (shape) veremos que son de 50 x 50 y por 3, esté se refiere a los canales de color (RGB) que tienen valores de 0 a 255.

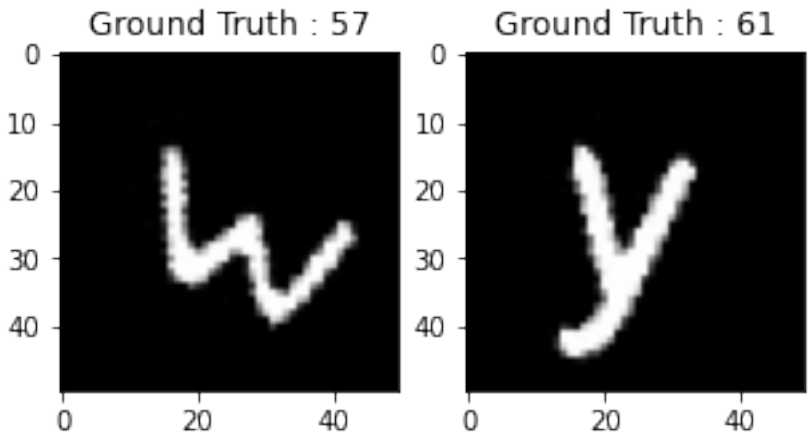
[7]:
```
train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.2)
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
Training data shape :  (334643, 50, 50, 3) (334643,)
Testing data shape :  (83661, 50, 50, 3) (83661,)
```

[8]:
```
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Display the first image in testing data
plt.subplot(122)
```

```
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

[8]:
```
Text(0.5, 1.0, 'Ground Truth : 61')
```

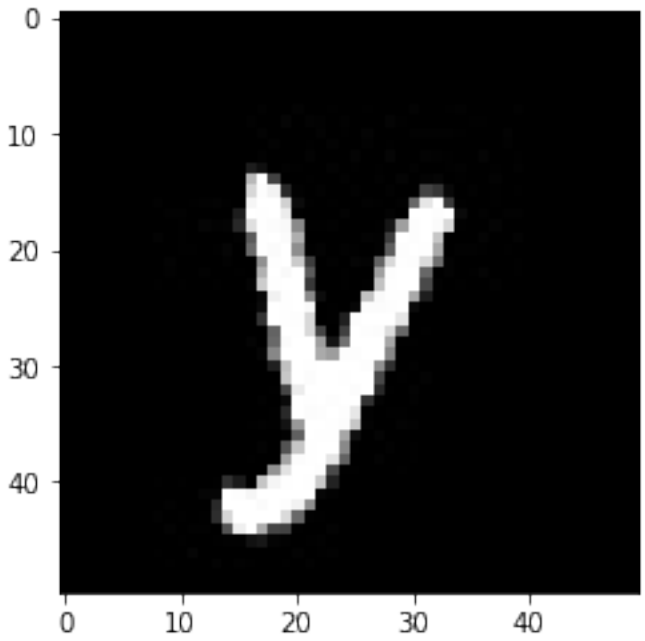


3.5 Preprocesamos las imagenes

Realizamos la normalización de valores para que tengan valor entre 0 y 1, por eso se realiza la división entre 255

[9]:
```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X/255.
test_X = test_X/255.
plt.imshow(test_X[0,:,:])
```

[9]:
```
<matplotlib.image.AxesImage at 0x12affdd9ee0>
```



3.6 Hacemos el One-hot Encoding para la red

El "One-Hot Encoding" nos ayuda a realizar la clasificación para la red neuronal y corresponde con una capa de salida de la red de 10 neuronas, esto se realiza con "to categorical".

[10]:
```
# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Display the change for category label using one-hot encoding
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 57
After conversion to one-hot: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

3.7 Creamos el Set de Entrenamiento y Validación

Subdividimos los datos en 80-20 para test y entrenamiento respectivamente. Y con el de training lo subdividimos en 80-20 para obtener un subconjunto de validación.

[11]:
```
#Mezclar todo y crear los grupos de entrenamiento y testing
train_X,valid_X,train_label,valid_label = train_test_split(train_X,␣
↪train_Y_one_hot, test_size=0.2, random_state=13)
```

[12]:
```
print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)
```

```
(267714, 50, 50, 3) (66929, 50, 50, 3) (267714, 64) (66929, 64)
```

3.8 Creamos el modelo de CNN

Aqui se crea la red, nos apoyamos de Keras para crear la CNN.

Las tres primeras constantes:

INIT_LR: Valor inicial de learning rate el 1e-1 corresponde al 0.1 Epochs: Cantidad de iteraciones completadas al conjunto de imagenes de entrenamiento. Batch_size: Cantidad de imágenes a procesar a la vez.

[13]:
```
#declaramos variables con los parámetros de configuración de la red
INIT_LR = 1e-1 # Valor inicial de learning rate. El valor 1e-3 coOrresponde con␣
↪0.001\ 0.1
epochs = 10 # Cantidad de iteraciones completas al conjunto de imagenes de␣
↪entrenamiento
batch_size = 64 # cantidad de imágenes que se toman a la vez en memoria
```

- Se crea una primera capa de neuronas convolucional de dos dimensiones en Cov2D, donde la entrada son las imagenes de 50x50x3
- Se aplican 32 filtros kernel 3x3
- La función de activacion para la neurona es “linear”
- MaxPooling(2,2) reduce la imagen al 50% es decir quedarán de 25x25
- Se termina con una capa de salida con 10 neurona con activacion SoftMax, que corresponde con el “Hot Encoding”

[1]:
```
letras_model = Sequential()
letras_model.add(Conv2D(32, kernel_size=(3,␣
↪3),activation='linear',padding='same',input_shape=(50,50,3)))#CAMBIAR TAMAÑO␣
↪DE IMAGEN
letras_model.add(LeakyReLU(alpha=0.1))
letras_model.add(MaxPooling2D((2, 2),padding='same'))
letras_model.add(Dropout(0.5))

letras_model.add(Flatten())
letras_model.add(Dense(32, activation='linear'))
letras_model.add(LeakyReLU(alpha=0.1))
letras_model.add(Dropout(0.5))
letras_model.add(Dense(nClasses, activation='softmax'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11104/3721401598.py in <module>
----> 1 letras_model = Sequential()
```

```
      2 letras_model.add(Conv2D(32, kernel_size=(3,␣
↪3),activation='linear',padding='same',input_shape=(50,50,3)))#CAMBIAR TAMAÑO␣
↪DE IMAGEN
      3 letras_model.add(LeakyReLU(alpha=0.1))
      4 letras_model.add(MaxPooling2D((2, 2),padding='same'))
      5 letras_model.add(Dropout(0.5))

NameError: name 'Sequential' is not defined
```

Error generado por ejecución equivoca, no genera salida visible.

[15]:
```
letras_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 32)        896

 module_wrapper (ModuleWrapp  (None, 50, 50, 32)       0
 er)

 max_pooling2d (MaxPooling2D  (None, 25, 25, 32)       0
 )

 dropout (Dropout)           (None, 25, 25, 32)        0

 flatten (Flatten)           (None, 20000)             0

 dense (Dense)               (None, 32)                640032

 module_wrapper_1 (ModuleWra  (None, 32)               0
 pper)

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 64)                2112

=================================================================
Total params: 643,040
Trainable params: 643,040
Non-trainable params: 0
_________________________________________________________________
```

Compilamos y asignamos optimizador, Adagrad es nuestro optimizador para este caso

[16]:
```
```

```
letras_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=tf.
↪keras.optimizers.Adagrad(learning_rate=INIT_LR, decay=INIT_LR /␣
↪100),metrics=['accuracy'])
```

3.9 Entrenamos el modelo: Aprende a clasificar imágenes

Entenamiento de la CNN, con la linea "letras_train" se inicia el entrenamiento y validación de nuestra red.

Este paso puede tomar varios minutos, dependiendo de tu ordenador, cpu y memoria ram libre.

[17]:
```
letras_train = letras_model.fit(train_X, train_label,␣
↪batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,␣
↪valid_label))
```

```
Epoch 1/10
4184/4184 [==============================] - 328s 78ms/step - loss: 1.6128 -
accuracy: 0.5148 - val_loss: 0.5075 - val_accuracy: 0.8898
Epoch 2/10
4184/4184 [==============================] - 328s 78ms/step - loss: 1.1055 -
accuracy: 0.6459 - val_loss: 0.3920 - val_accuracy: 0.9234
Epoch 3/10
4184/4184 [==============================] - 332s 79ms/step - loss: 1.0189 -
accuracy: 0.6736 - val_loss: 0.3503 - val_accuracy: 0.9316
Epoch 4/10
4184/4184 [==============================] - 327s 78ms/step - loss: 0.9789 -
accuracy: 0.6858 - val_loss: 0.3270 - val_accuracy: 0.9369
Epoch 5/10
4184/4184 [==============================] - 331s 79ms/step - loss: 0.9585 -
accuracy: 0.6916 - val_loss: 0.3144 - val_accuracy: 0.9393
Epoch 6/10
4184/4184 [==============================] - 326s 78ms/step - loss: 0.9370 -
accuracy: 0.6965 - val_loss: 0.3043 - val_accuracy: 0.9427
Epoch 7/10
4184/4184 [==============================] - 330s 79ms/step - loss: 0.9260 -
accuracy: 0.7008 - val_loss: 0.2970 - val_accuracy: 0.9440
Epoch 8/10
4184/4184 [==============================] - 332s 79ms/step - loss: 0.9199 -
accuracy: 0.7023 - val_loss: 0.2920 - val_accuracy: 0.9449
Epoch 9/10
4184/4184 [==============================] - 331s 79ms/step - loss: 0.9139 -
accuracy: 0.7038 - val_loss: 0.2871 - val_accuracy: 0.9468
Epoch 10/10
4184/4184 [==============================] - 334s 80ms/step - loss: 0.9052 -
accuracy: 0.7071 - val_loss: 0.2857 - val_accuracy: 0.9472
```

Guardamos la red, para reutilizarla en el futuro, sin tener que volver a entrenar

Vemos que tras 10 iteraciones completas al set de entrenamiento obtenemos un valor de presición de 70.71% y en el set de validación alcanza un 94.72%

[18]:
```
# guardamos la red, para reutilizarla en el futuro, sin tener que volver a␣
↪entrenar
letras_model.save("C:\\Users\\gilba\\Desktop\\IA\\letras94.h5py")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
leaky_re_lu_layer_call_fn, leaky_re_lu_layer_call_and_return_conditional_losses,
leaky_re_lu_1_layer_call_fn,
leaky_re_lu_1_layer_call_and_return_conditional_losses while saving (showing 5
of 5). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to:
C:\Users\gilba\Desktop\IA\letras94.h5py\assets

INFO:tensorflow:Assets written to:
C:\Users\gilba\Desktop\IA\letras94.h5py\assets
```

3.10 Evaluamos la red

Ya con la red entrenada, la ponemos a prueba con el set de imágenes para Test que separamos al principio, cabe resaltar que son muestras que nunca han sido "vistas" por la máquina.

[19]:
```
test_eval = letras_model.evaluate(test_X, test_Y_one_hot, verbose=1)
```

```
2615/2615 [==============================] - 37s 14ms/step - loss: 0.2872 -
accuracy: 0.9467
```

[20]:
```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.28720563650131226
Test accuracy: 0.9466537833213806
```

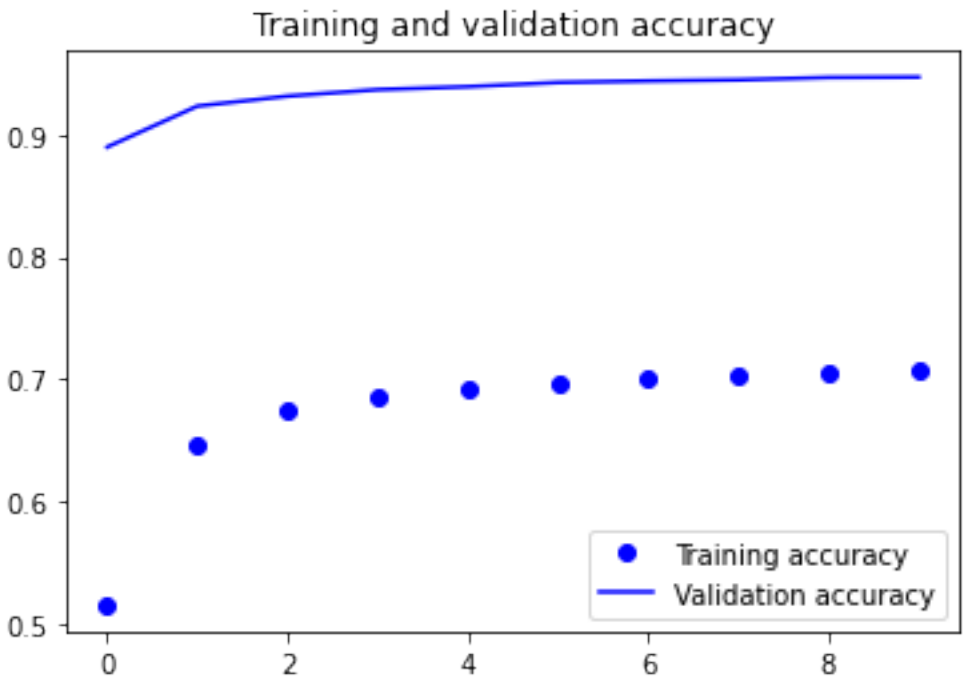
Vemor que el conjunto de test alcanza una presición del 94.66%

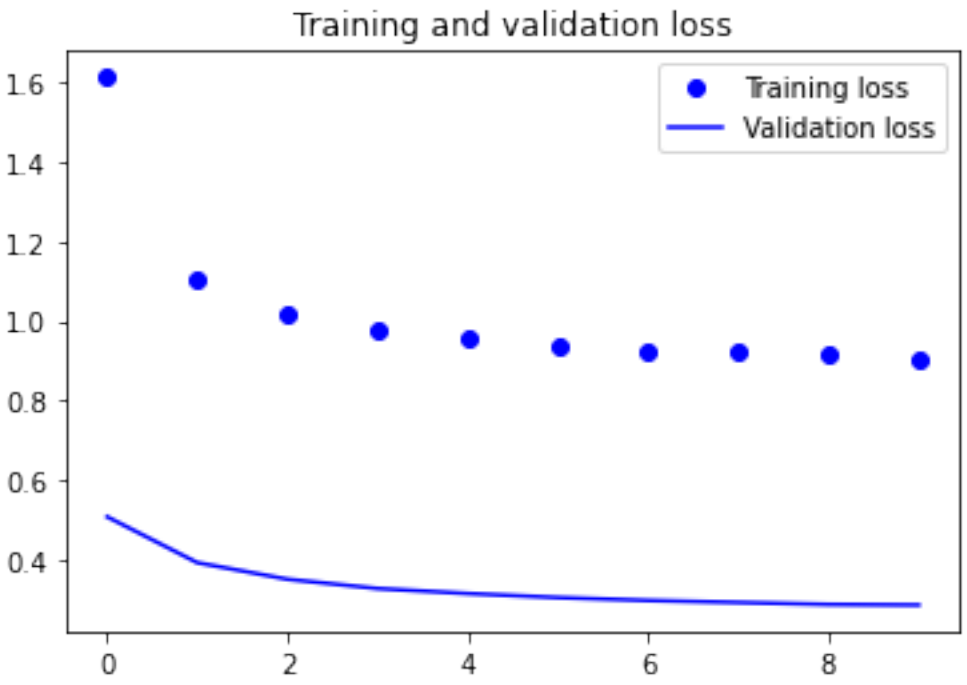
[21]:
```
letras_train.history
```

[21]:
```
{'loss': [1.6128021478652954,
  1.105499505996704,
  1.018921136856079,
  0.9789007306098938,
  0.9584579467773438,
  0.9369893670082092,
  0.9260150790214539,
  0.9199399948120117,
  0.9138894081115723,
  0.9052349925041199],
 'accuracy': [0.5147956609725952,
  0.6459206342697144,
  0.6736330389976501,
  0.6857765913009644,
  0.6916074752807617,
```

```
 0.6965455412864685,
 0.7008113265037537,
 0.7022830247879028,
 0.7038257122039795,
 0.7070829272270203],
'val_loss': [0.5074700713157654,
 0.3920218050479889,
 0.35033318400382996,
 0.32699212431907654,
 0.31440815329551697,
 0.30426889657974243,
 0.29700037837028503,
 0.2919713854789734,
 0.28705692291259766,
 0.2856680750846863],
'val_accuracy': [0.889838457107544,
 0.9233964085578918,
 0.9315692782402039,
 0.9368883371353149,
 0.9393088221549988,
 0.9427452683448792,
 0.9440153241157532,
 0.9448968172073364,
 0.9468392133712769,
 0.9471529722213745]}
```

[22]:
```
accuracy = letras_train.history['accuracy']
val_accuracy = letras_train.history['val_accuracy']
loss = letras_train.history['loss']
val_loss = letras_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





[23]:
```
predicted_classes2 = letras_model.predict(test_X)
```

```
2615/2615 [==============================] - 34s 13ms/step
```

[24]:
```
predicted_classes=[]
for predicted_letras in predicted_classes2:
    predicted_classes.append(predicted_letras.tolist().
 ↪index(max(predicted_letras)))
predicted_classes=np.array(predicted_classes)
```

[25]:
```
predicted_classes.shape, test_Y.shape
```

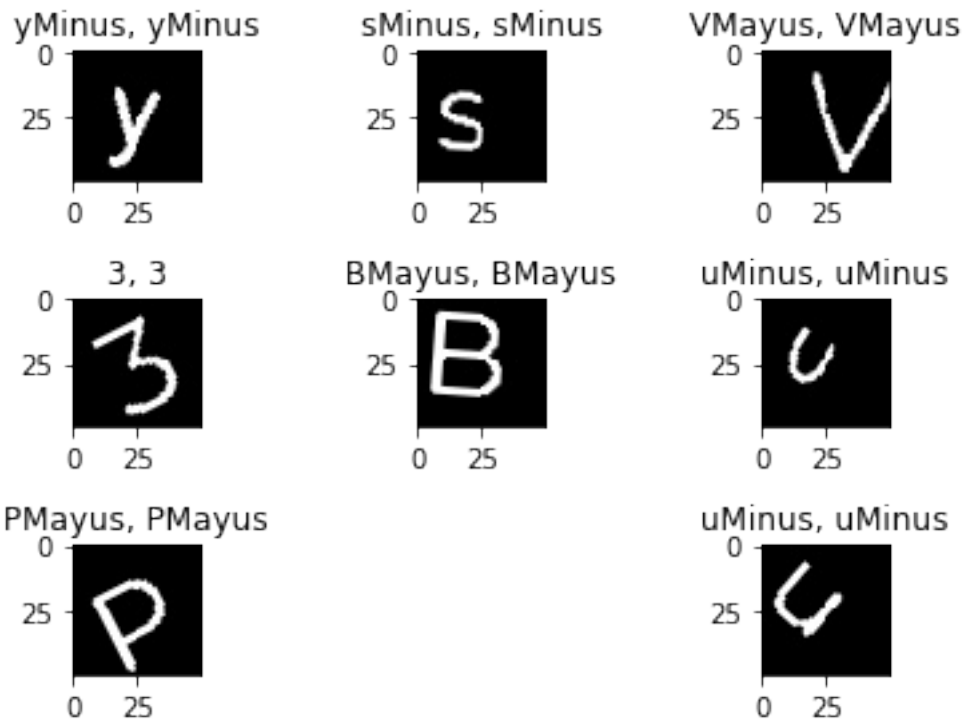
[25]:
```
((83661,), (83661,))
```

3.11 Aprendamos de los errores: Qué mejorar

Realizamos un analisis más profundo para darnos cuenta de que mejorar y como realizarlo, ya sea mejorando nuestro data set o dandole mas ejemplos a la neurona para que así logre un mejor aprendizaje del objeto, panorama, acción, etc. a detectar.

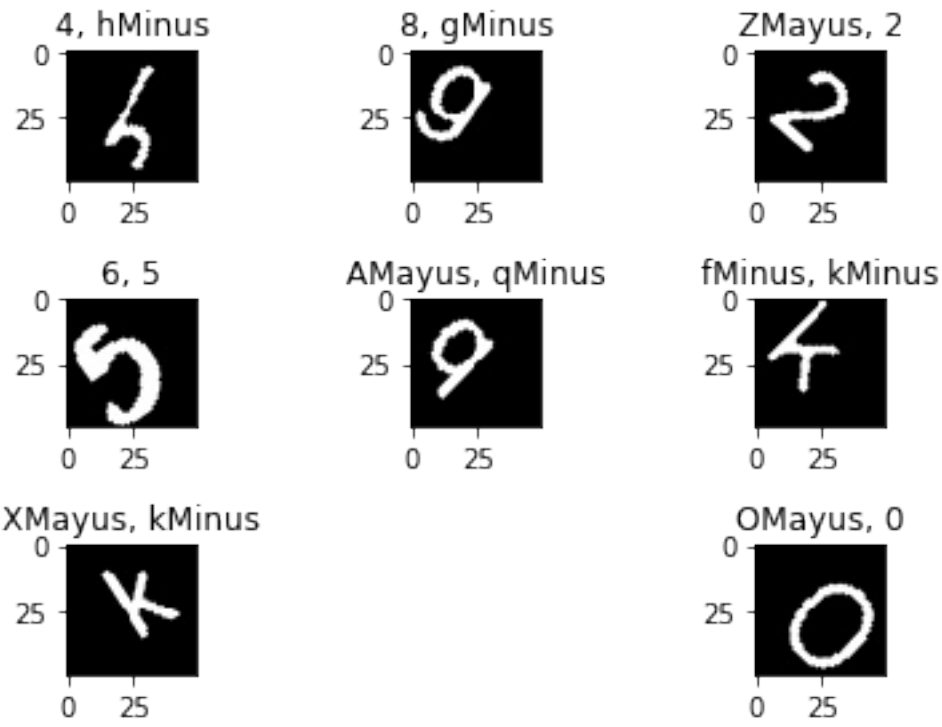
[26]:
```
correct = np.where(predicted_classes==test_Y)[0]
print("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[0:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[correct].reshape(50,50,3), cmap='gray',␣
 ↪interpolation='none')#CAMBIAR TAMAÑO DE IMAGEN
    plt.title("{}, {}".format(letras[predicted_classes[correct]],
                                                    letras[test_Y[correct]]))

    plt.tight_layout()
```

```
Found 79198 correct labels
```



[27]:
```
incorrect = np.where(predicted_classes!=test_Y)[0]
print("Found %d incorrect labels" % len(incorrect))
for i, incorrect in enumerate(incorrect[0:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[incorrect].reshape(50,50,3), cmap='gray',␣
 ↪interpolation='none')
    plt.title("{}, {}".format(letras[predicted_classes[incorrect]],
                                                    letras[test_Y[incorrect]]))
    plt.tight_layout()
```

```
Found 4463 incorrect labels
```



[28]:
```
target_names = ["Class {}".format(i) for i in range(nClasses)]
print(classification_report(test_Y, predicted_classes,␣
 ↪target_names=target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.91      0.83      0.87      1322
     Class 1       0.90      0.95      0.93      1311
     Class 2       0.97      0.93      0.95      1283
     Class 3       0.98      0.94      0.96      1361
     Class 4       0.94      0.92      0.93      1243
     Class 5       0.95      0.94      0.95      1319
     Class 6       0.96      0.96      0.96      1277
     Class 7       0.93      0.94      0.94      1329
     Class 8       0.85      0.98      0.91      1343
     Class 9       0.87      0.84      0.85      1355
    Class 10       0.97      0.98      0.98      1295
    Class 11       0.97      0.95      0.96      1299
    Class 12       0.96      1.00      0.98      1305
    Class 13       0.98      0.96      0.97      1307
    Class 14       1.00      0.96      0.98      1334
    Class 15       0.98      0.97      0.97      1288
    Class 16       0.91      0.98      0.95      1291
    Class 17       0.97      0.94      0.96      1291
```

Class 18	0.98	0.99	0.98	1321
Class 19	0.97	0.96	0.97	1265
Class 20	0.97	0.98	0.98	1243
Class 21	0.94	0.95	0.94	1244
Class 22	0.94	0.97	0.96	1306
Class 23	0.89	0.91	0.90	1313
Class 24	0.88	0.99	0.93	1251
Class 25	0.98	0.90	0.94	1275
Class 26	0.85	0.89	0.87	1330
Class 27	0.90	0.97	0.93	1290
Class 28	0.98	0.94	0.96	1384
Class 29	0.98	0.92	0.95	1311
Class 30	0.98	0.98	0.98	1284
Class 31	0.94	0.95	0.95	1282
Class 32	0.99	0.98	0.98	1322
Class 33	0.75	0.99	0.85	1289
Class 34	0.95	0.99	0.97	1316
Class 35	0.99	0.97	0.98	1320
Class 36	0.99	0.83	0.90	1347
Class 37	0.99	0.99	0.99	1295
Class 38	0.98	0.99	0.98	1319
Class 39	0.96	0.99	0.97	1295
Class 40	0.91	0.90	0.90	1288
Class 41	0.95	0.99	0.97	1370
Class 42	0.97	0.94	0.95	1246
Class 43	0.93	0.90	0.92	1322
Class 44	0.93	0.94	0.93	1325
Class 45	0.94	0.93	0.93	1321
Class 46	0.96	1.00	0.98	1337
Class 47	0.95	0.96	0.96	1295
Class 48	0.98	0.92	0.95	1297
Class 49	0.98	0.97	0.97	1292
Class 50	0.99	0.98	0.99	1343
Class 51	0.92	0.71	0.80	1302
Class 52	1.00	0.98	0.99	1353
Class 53	0.99	0.96	0.97	1344
Class 54	0.94	0.97	0.96	1309
Class 55	0.95	0.97	0.96	1300
Class 56	0.96	0.97	0.97	1288
Class 57	0.99	0.97	0.98	1311
Class 58	0.95	0.98	0.97	1314
Class 59	0.97	0.98	0.98	1291
Class 60	0.88	0.97	0.93	1317
Class 61	0.93	0.78	0.85	1357
Class 62	0.94	0.97	0.95	1301
Class 63	0.99	0.97	0.98	1283
accuracy			0.95	83661

```
   macro avg       0.95      0.95      0.95     83661
weighted avg       0.95      0.95      0.95     83661
```

Podemos observar que en la presición final de un analisis más profundo tenemos un acercamiento al reconocimiento, por ejemplo, de '0' del 91%, de 'A' del 97%, de 'Z' del 94% y por ultimo ejemplo de '8' del 85%.

Dandonos un promedio de presición del 95%

4 Aplicación del Módelo, Validación

Para la aplicación (Validación) del Modelo, haremos uso del archivo guardado "letras94.hp5py".

El código no es extenso, ni dificil de comprender pero decidi comentarlo en el mismo para mayor conprensión y poder darme a explicar más conforme a las funciones y parametros utilizados.

4.1 Importación de Librerías

[2]:
```
import numpy as np
from keras import models
import cv2 as cv
```

4.2 Cargamos el Módelo

[3]:
```
model = models.load_model("C:
↪\\Users\\gilba\\Desktop\\IA\\ia\\ProyectoFinal\\Letras\\letras94.h5py")
```

4.3 Aseguramos la carga del Módelo

La impresión que nos genere el "summary" debe ser identica al que generamos al crear el Modelo en la Red Convolucional

[4]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 32)        896

 module_wrapper (ModuleWrapp  (None, 50, 50, 32)       0
 er)

 max_pooling2d (MaxPooling2D  (None, 25, 25, 32)       0
 )

 dropout (Dropout)           (None, 25, 25, 32)        0
```

```
 flatten (Flatten)           (None, 20000)             0

 dense (Dense)               (None, 32)                640032

 module_wrapper_1 (ModuleWra  (None, 32)               0
 pper)

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 64)                2112

=================================================================
Total params: 643,040
Trainable params: 643,040
Non-trainable params: 0
_________________________________________________________________
```

4.4 Arreglo de Clases

[5]:
```
clases_letras = ['0', '1', '2','3','4'␣
 ↪,'5','6','7','8','9','AMayus','aMinus','BMayus','bMinus','CMayus','cMinus','DMayus',
              'dMinus','EMayus','eMinus','FMayus','fMinus','GMayus','gMinus',␣
 ↪'HMayus', 'hMinus', 'IMayus', 'iMinus',
              'JMayus', 'jMinus', 'KMayus','kMinus', 'LMayus', 'lMinus',␣
 ↪'MMayus', 'mMinus', 'NMayus', 'nMinus', 'NNMayus',
              'nnMinus', 'OMayus', 'oMinus','PMayus', 'pMinus', 'QMayus',␣
 ↪'qMinus', 'RMayus', 'rMinus', 'SMayus', 'sMinus',
              'TMayus', 'tMinus', 'UMayus', 'uMinus', 'VMayus', 'vMinus',␣
 ↪'WMayus', 'wMinus','XMayus','xMinus', 'YMayus',
              'yMinus','ZMayus','zMinus']

#DECALRAMOS UN CONTADOR PARA NO DESBORDAR EL CICLO DE BUSQUEDA DE LA␣
 ↪PREDICCION,
#SOBRE TODO PARA QUE SE VEA MAS CLARO, VISUAL
contador = len(clases_letras)
```

4.5 Tratamiento de Imagen, Np Array

[6]:
```
def tratarImg(im):
    #SE PODRIAN AGREGAR FILTRO DE COLOR Y REDIMENSION PARA TRATAR LA IMAGEN␣
 ↪ANTES DE
    #CONVERTIRLA A UN ARRAY Y REALIZAR LA PREDICCION, ESTO NOS DARIA MAYOR␣
 ↪PRESICION


    #SE CORRIJE LA GAMA DE LA IMAGEN DE BGR A RGB :)
    im = cv.cvtColor(im, cv.COLOR_GRAY2BGR)
```

```
    cv.imshow("imagen2", im)
    #SE CREA EL NUMPY ARRAY
    img_array = np.array(im)

    #EXPANDE LA MATRIZ ARRAY PARA EVITAR EL ERROR NONE,50,50,3
    #PRIMER VALOR NUMPY ARRAY
    #SEGUNDO VALOR AXIS, POSICION
    img_array = np.expand_dims(img_array, axis=0)

    #RETORNAMOS EL NUMPY ARRAY DE LA IMAGEN
    return img_array
```

4.6 Realizamos la Predicción

[ ]:
```
#LEEMOS LA IMAGEN
#IM LA USAREMOS PARA REALIZAR LA PREDICCION
#PRUEBA13 8 MINUS CORRECTA
#PRUEBA16 g MINUS CORRECTA
#PRUEBA18 NN MINUS CORRECTA
#PRUEBA19 NN MINUS CORRECTA
im = cv.imread('C:
 ↪\\Users\\gilba\\Desktop\\IA\\ia\\ProyectoFinal\\Letras\\AbecedariosPruebas\\Prueba21.
 ↪jpg', 0)

#IM2 LA USAREMOS PARA MOSTRARLA UNICAMENTE
imagen2 = cv.imread('C:
 ↪\\Users\\gilba\\Desktop\\IA\\ia\\ProyectoFinal\\Letras\\AbecedariosPruebas\\Prueba21.
 ↪jpg')
#RESIZE PARA VER EL RESULTADO MAS AMPLIO
imagen2 = cv.resize(imagen2,(600,600))

#INICIALIZACION DE MOSTRAR, NO ES NECESARIA PERO SI REQUERIDA
mostrar = ""

#SE ENVIA IM AL METODO PARA TRATARLA Y CONVERTIRLA EN ARRAY
img_array = tratarImg(im)

#SE REALIZA LA PREDICCION CON EL MODELO Y LA IMAGEN ARRAY
prediction = model.predict(img_array)

#SE IMPRIME LA PREDICCION SOLO PARA VALIDAR EL RESULTADO GRAFICO
print (prediction)

#RECORREMOS EL ARREGLO Y GUARDAMOS LA PREDICCION PARA AGREGARLA A LA IMAGEN␣
 ↪FINAL
for i in range(contador):
```

```
    #SI EL VALOR DE LA CASILLA [0][i] ES IGUAL A 1 SACAMOS EL NOMBRE DE ESA␣
 ↪MISMA CELDA EN EL ARRAY CLASES
    if prediction[0][i].all() == 1 :
        #GUARDA LA PREDICCION PARA MOSTRAR DICHO TEXTO EN LA IMAGEN
        mostrar="Predice: "+clases_letras[i]
        #ROMPE EL CICLO SI ENCONTRO UN 1
        break

#ELIMINAMOS RUIDO DE LA IMAGEN CON FUNCION GAUSSIANBLUR
#PRIMER PARAMETRO IMAGEN A LIMPIAR
#SEGUNDO PARAMETRO TAMANIO DE KERNEL, EN ESTE CASO 5 X 5 PX SOLO QUEREMOS␣
 ↪ELIMIANR RUIDO NO QUEREMOS VERLA BORROSA
#TERCER PARAMETEO ESPECIFICA LOS LIMITES DE LA IMAGEN
gaussiana = cv.GaussianBlur(imagen2, (5,5), 0)

#CONVERTIMOS DE RGB A GRISES
#PRIMER PARAMETRO IMAGEN A CONVERTIR
#SEGUNDO PARAMETRO DE QUE ESCALA A QUE ESCALA
imageOut = cv.cvtColor(gaussiana, cv.COLOR_BGR2GRAY)

#BINARISAMOS LA IMAGEN PARA VERLA MAS CLARA
for x in range(imageOut.shape[0]):
    for y in range(imageOut.shape[1]):
        if imageOut[x,y] < 150:
            imagen2[x,y]=0 #PONE EL PIXEL EN NEGRO
        else:
            imagen2[x,y]=255 #PONE EL PIXEL EN BLANCO

#PUT TEXT AGREGA EL TEXTO EN LA IMAGEN
#PRIMER PARAMETRO IMAGEN A MOSTRAR
#SEGUNDO PARAMETRO TEXTO A AGREGAR
#TERCER PARAMETRO POSICION
#CUARTO PARAMETRO TIPO DE LETRA
#QUINTO PARAMETRO TAMANIO DE LA LETRA
#SEXTO PARAMETRO COLOR
#SEPTIMO PARAMETRO TRAZO DE LA LETRA, NUMERO DE PIXELES
cv.putText(imagen2, mostrar,(0,50), cv.FONT_HERSHEY_SIMPLEX, 1,(50,255,0),2)

#RECTANGLE AGREGA EL TEXTO EN LA IMAGEN
#PRIMER PARAMETRO IMAGEN A MOSTRAR
#SEGUNDO PARAMETRO INICIO DEL RECTANGULO
#TERCER PARAMETRO FIN DEL RECTANGULO
#CUARTO PARAMETRO TIPO DE LETRA
#QUINTO PARAMETRO COLOR
#SEXTO PARAMETRO TRAZO DEL RECTANGULO
cv.rectangle(imagen2, (100, 100), (500, 550) , (255, 0, 0) , 2)
```

```
cv.imshow("imagen", imagen2)
cv.waitKey(0)
cv.destroyAllWindows()
```

No se muestra el entorno gráfico en la predicción pero usted puede ejecutar todo el código en esté reporte y en esta ultima instancia le generará un entorno gráfico donde muestra la imágen a predecir y la predicción dada por la CNN.