**UNIVERSIDAD PRIVADA FRANZ TAMAYO**

**FACULTAD DE INGENIERIA**

**CARRERA DE INGENIERIA DE SISTEMAS**

****

**¨beating captchas with neural networks¨**

Integrantes:

Jomar Camacho

Sandra Maldonado

Gabriela Valencia

Joel Arancibia murillo

Docente: Miguel Ángel Paco

Cochabamba-Bolivia

# Introducción. –

Este programa que se llegó a desarrollar, tiene como uso detectar fraudes por duplicación en la escritura, en otras palabras con el desarrollo de este software podremos detectar el duplicado de firmas de identidad o hasta vulnerar el captcha de las páginas web todo esto gracias al tema beating captchas aprendida en minería datos creando una pequeña inteligencia artificial que aprenderá el abecedario y reconocer las letras luego de mostrárselas así como respuesta escribir el mismo texto burlando los captchas de seguridad.

# Objetivos. –

* Desarrollar una inteligencia artificial que logre aprender el abecedario para poder reconocer las letras
* Aplicar los conocimientos aprendidos en los libros sobre el beating captchas
* Aplicar conocimientos previos acerca del lenguaje Python
* Realizar una investigacion acerca de las librerías para el uso de reconocimiento de letras

1. **Marco teórico. –**

**import** **numpy** **as** **npn //instalamos las librerías necesarias**

**import** **pybrain**

**from** **PIL** **import** Image, ImageDraw, ImageFont

**from** **skimage** **import** transform **as** tf

%matplotlib inline

**from** **matplotlib** **import** pyplot **as** plt

Mediante esto importamos las herramientas necesarias de cada una de las librerías que necesitamos para el desarrollo

**from** **skimage.measure** **import** label, regionprops

**from** **sklearn.utils** **import** check\_random\_state

random\_state = check\_random\_state(14)

letters = list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

shear\_values = np.arange(0,0.5,0.05)

**from** **sklearn.preprocessing** **import** OneHotEncoder

onehot = OneHotEncoder()

**from** **skimage.transform** **import** resize

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **pybrain.datasets** **import** SupervisedDataSet

**from** **pybrain.tools.shortcuts** **import** buildNetwork

**from** **pybrain.supervised.trainers** **import** BackpropTrainer

**from** **sklearn.metrics** **import** f1\_score

CREATE CAPTCHA

**def** create\_captcha(text, shear=0,size=(100,24)):/definimos la función para la creación del captcha con los parámetros que deseamos

im = Image.new("L", size, "black")// la imagen que tendrá el modelo”L” tamaño y color como parametros

draw = ImageDraw.Draw(im)// se dibuja la imagen ya establecida con los parámetros

font = ImageFont.truetype(r"Coval.otf", 22)//colocamos un tipo de letra

draw.text((2, 2), text, fill=1, font=font)

image = np.array(im)

affine\_tf = tf.AffineTransform(shear=shear)

image = tf.warp(image, affine\_tf)

**return** image/image.max()

image = create\_captcha("GENE",shear=0.5)//generamos un captcha con un texto definido

plt.imshow(image,cmap='Greys')//lo mostramos como grafica con el plt.imshow

<matplotlib.image.AxesImage at 0x16cba422f98>



Resultado o salida

SEGMENT CAPTCHA

**def** segment\_image(image)://definimos la función con la que segmentaremos la imagen captcha

labeled\_image = label(image>0)

subimages = []//definimos una variable tipo array para las subimagenes

**for** region **in** regionprops(labeled\_image):

start\_x, start\_y, end\_x, en\_y = region.bbox

subimages.append(image[start\_x:end\_x,start\_y:en\_y])

**if** len(subimages) == 0:

División de captcha por segmentos

**return** [image,]

**return** subimages

subimages = segment\_image(image)

f, axes = plt.subplots(1, len(subimages),figsize=(10,3))

**for** i **in** range(len(subimages)):

axes[i].imshow(subimages[i], cmap="gray")



DATASET RECONIZE EACH LETTER

**def** generate\_sample(random\_state=**None**):

random\_state = check\_random\_state(random\_state)

letter = random\_state.choice(letters)

shear = random\_state.choice(shear\_values)

**return** create\_captcha(letter,shear=shear,size=(30,30)),letters.index(letter)

image, target = generate\_sample(random\_state)

plt.imshow(image,cmap="Greys")

print("The target for this image is: **{0}**".format(target))

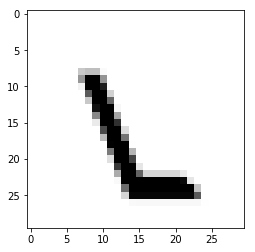
dataset, targets = zip(\*(generate\_sample(random\_state)**for** i **in** range(3000)))

Reconocimiento por letra

dataset= np.array(dataset, dtype='float')

targets = np.array(targets)

The target for this image is: 11



PREDICTING WORDS

subimages = segment\_image(captcha\_image)

predicted\_word = ""

**for** subimage **in** subimages:

subimage = resize(subimage, (20, 20))

outputs = net.activate(subimage.flatten())

prediction = np.argmax(outputs)

predicted\_word += letters[prediction]

**return** predicted\_word

word = "GENE"

prediction

captcha = create\_captcha(word, shear=0.2)

print(predict\_captcha(captcha, net))

**def** test\_prediction(word, net, shear=0.2):

captcha = create\_captcha(word, shear=shear)

prediction = predict\_captcha(captcha, net)

**return** word == prediction, word, prediction

**from** **nltk.corpus** **import** words

Importación de librería nltk para descarga de package words

**import** **nltk**

nltk.download('words')

[nltk\_data] Downloading package words to

[nltk\_data] C:\Users\HP\AppData\Roaming\nltk\_data...

[nltk\_data] Package words is already up-to-date!

valid\_words = set([word.upper() **for** word **in** words.words() **if** len(word) == 4])

num\_correct = 0

num\_incorrect = 0

**for** word **in** valid\_words:

correct, word, prediction = test\_prediction(word, net, shear=0.2)

**if** correct:

num\_correct += 1

**else**:

num\_incorrect += 1

print("Number correct is **{0}**".format(num\_correct))

print("Number incorrect is **{0}**".format(num\_incorrect))

Number correct is 0

Number incorrect is 4995

**from** **sklearn.metrics** **import** confusion\_matrix

cm = confusion\_matrix(np.argmax(y\_test, axis=1), predictions)

plt.figure(figsize=(10, 10))

plt.imshow(cm)

tick\_marks = np.arange(len(letters))

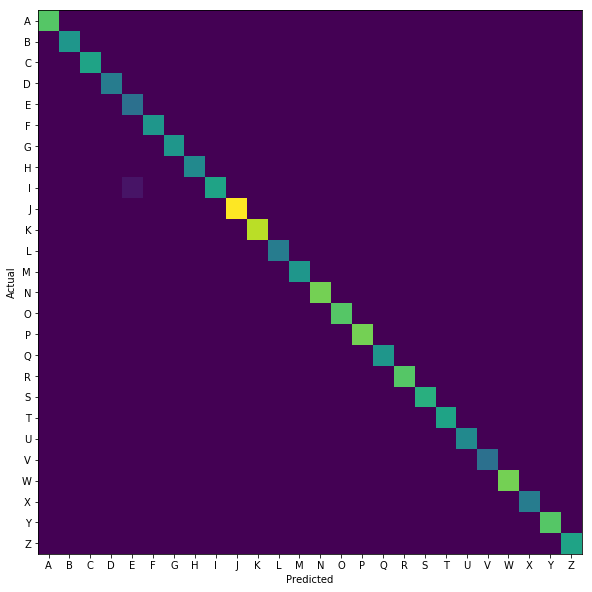
plt.xticks(tick\_marks, letters)

plt.yticks(tick\_marks, letters)

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()



PUTTING IT ALL TOGETHER

**from** **operator** **import** itemgetter

**def** improved\_prediction(word, net, dictionary, shear=0.2):

captcha = create\_captcha(word, shear=shear)

prediction = predict\_captcha(captcha, net)

**if** prediction **not** **in** dictionary:

distances = sorted([(word, compute\_distance(prediction, word)) **for** word **in** dictionary],

key=itemgetter(1))

best\_word = distances[0]

prediction = best\_word[0]

**return** word == prediction, word, prediction

num\_correct = 0

num\_incorrect = 0

**for** word **in** valid\_words:

correct, word, prediction = improved\_prediction(word, net, valid\_words, shear=0.2)

**if** correct:

num\_correct += 1

**else**:

num\_incorrect += 1

print("Number correct is **{0}**".format(num\_correct))

print("Number incorrect is **{0}**".format(num\_incorrect))