**UNIVERSIDAD PRIVADA FRANZ TAMAYO**

**FACULTAD DE INGENIERIA**

**CARRERA DE INGENIERIA DE SISTEMAS**



**“Captcha”**

Estudiante:

* Erick Rodrigo Aranibar Arias
* Einar David Villarroel Vargas
* Rene Rocha Antezana

FECHA: 6/12/2019

MATERIA: Minería de Datos

Cochabamba – Bolivia

2019

1. **Introducción**

La interpretación de imágenes siempre fue un problema difícil de resolver, pero los últimos avances han permitido detectar y comprender imágenes hasta el punto de automatizar un sistema de vigilancia, teniendo la capacidad de reconocimiento facial.

Es un proceso bastante difícil, debido a la calidad y claridad de las imágenes.

Se desarrollará la resolución automática del CAPTCHA, que son imágenes que el ojo humano las resuelve fácilmente pero un ordenador no, impidiendo así el uso de programas automatizados usados en cuentas falsas o mensajes Spam.

Se llevará a cabo el uso de redes neuronales, la creación propia de un conjunto de CAPTCHA, letras y las librerías scikit y PyBrain para el trabajo de imágenes y redes neuronales conjuntas en Python.

1. **Desarrollo**

Realizar la creación de un programa que sea capaz de que pueda vencer el CAPTCHA.

* 1. **Librerías**
  + Scikit
  + PyBrain
  1. **Pasos:**
  + Dividir la imagen en sus letras individuales
  + clasificar las letras
  + recombinar las letras para formar una nueva palabra
  + clasificar las palabras con un diccionario para tratar de corregir errores.
  + dibujar una imagen y realizar un corte para almacenar los fragmentos en una matriz. (El tamaño de adaptación para cada imagen que variara ligeramente con cada letra generada)
  + luego mostrarla como cualquier grafica con la librería matplotlib.
  + Identificar cada una de las letras separadas. (Usar funciones para encontrar conjuntos de pixeles que den el mismo valor, siendo las imágenes no identificadas valoradas con un cero, para ello se deben identificar las regiones del objeto para obtener la región actual de la imagen, tomando como parámetro el origen y el fin de la región a evaluar).
  + Extraer las imágenes cortadas por partes (agregar la imagen a un arreglo simple para poder enseñar cada imagen fragmentada).
  + Crear una función para realizar un conjunto de datos de letras con diferentes valores, así la red neuronal pueda reconocer cada letra.
  + Mostrar el funcionamiento del conjunto de datos para luego devolver el valor de las letras junto con las imágenes. (Valores que oscilaran del 0 al 26, tratando de no exceder los valores que puede soportar cada neurona de la red neuronal se manejaran valores del 0 al 1 para identificar si la letra se parece o no).

Normalmente el conjunto de datos seria tri-dimensional, pero necesitaremos trabajar con un conjunto bi-dimensional, para construir una red neural que tomara una imagen como entrada y podrá predecir cada letra que se encuentre en la imagen.

Ahora se podrá construir una red neural de 3 capas, 400 neuronas compondrán la primera capa y 26 neuronas estarán listas para la salida de datos, pero determinar el número de neuronas en la capa oculta entre ambas, ya que demasiadas neuronas lo harían difícil de entrenar, por el momento se usarán 100 neuronas para la capa oculta.

Para corregir los posibles errores se usará el algoritmo de propagación inversa, que “culpará” a las neuronas por cada mala predicción, ajustando el peso hacia esas neuronas, este peso afectará a la taza de aprendizaje que será el parámetro del algoritmo (normalmente en un valor bajo).

Se calculará la gradiente del error de la función:

* multiplicar por la taza de aprendizaje
* restar el peso

según el error y el peso que se esté restando, este siempre tratara de corregirse a la dirección correcta, al menos hasta llegar a lo que se llamara optimo local (donde los pesos son similares pero el mejor conjunto de pesos). Este algoritmo retrocederá de capa en capa hasta llegar a la entrada de información, entonces los pesos de las conexiones neuronales serán actualizados.

El algoritmo de backdrop se ejecuta de manera iterativa usando los conjuntos de entrenamiento para ajustar el peso de las neuronas un poco, siendo factible el parar la ejecución cuando el error se reduce a una cantidad muy pequeña y pierde el sentido el entrenamiento.

Para evitar ese sentido de iteraciones, se definen “épocas” que determinaran la duración de la iteración del algoritmo, en este caso usaremos 20 épocas. Estas iteraciones darán como resultado una puntuación de 0.97, el cual es un gran resultado para un algoritmo neural tan simple como este.

* 1. **Código**

#%matplotlib inline

#from matplotlib import pyplot as plt

import numpy as np

from PIL import Image, ImageDraw, ImageFont

from skimage import transform as tf

def create\_captcha(text, shear=0, size=(100, 34)):

im = Image.new("L", size, "black")

draw = ImageDraw.Draw(im)

font = ImageFont.truetype(r"LiberationSans-Regular.ttf", 22)

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()

# from IPython import get\_ipython

# get\_ipython().run\_line\_magic('matplotlib', 'inline')

# matplotlib.use('TkAgg')

# get\_ipython().run\_line\_magic('matplotlib', 'osx')

# matplotlib.use('TkAgg')

# %matplotlib osx

import matplotlib.pyplot as plt

image = create\_captcha("ERICK", shear=0.5)

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

plt.savefig("foto.png")

from skimage.measure import label, regionprops

def segment\_image(image):

labeled\_image = label(image > 0)

subimages = []

for region in regionprops(labeled\_image):

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

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

if len(subimages) == 0:

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")

#axes[i].savefig("letra"+(i+1))

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)

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=(20, 20)), 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)))

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

targets = np.array(targets)

from sklearn.preprocessing import OneHotEncoder

onehot = OneHotEncoder()

y = onehot.fit\_transform(targets.reshape(targets.shape[0],1))

y = y.todense()

from skimage.transform import resize

dataset = np.array([resize(segment\_image(sample)[0], (20, 20)) for

sample in dataset])

X = dataset.reshape((dataset.shape[0], dataset.shape[1] \* dataset.

shape[2]))

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

X\_train, X\_test, y\_train, y\_test = \

train\_test\_split(X, y, train\_size=0.9)

plt.savefig("foto2.png")

# Desde acá no da.

from pybrain.datasets import SupervisedDataSet

training = SupervisedDataSet(X.shape[1], y.shape[1])

for i in range(X\_train.shape[0]):

training.addSample(X\_train[i], y\_train[i])

testing = SupervisedDataSet(X.shape[1], y.shape[1])

for i in range(X\_test.shape[0]):

testing.addSample(X\_test[i], y\_test[i])

from pybrain.tools.shortcuts import buildNetwork

net = buildNetwork(X.shape[1], 100, y.shape[1], bias=True)

from pybrain.supervised.trainers import BackpropTrainer

trainer = BackpropTrainer(net, training, learningrate=0.01,

weightdecay=0.01)

trainer.trainEpochs(epochs=20)

predictions = trainer.testOnClassData(dataset=testing)

from sklearn.metrics import f1\_score

print("F-score: {0:.2f}".format(f1\_score(predictions, y\_test.argmax(axis=1),average = 'micro' )))

def predict\_captcha(captcha\_image, neural\_network):

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 = "ERICK"

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)

prediction = prediction[:4]

return word == prediction, word, prediction

import nltk;

nltk.download('words')

from nltk.corpus import words

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

print('llega')

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))

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()

plt.savefig('foto3.png')

def compute\_distance(prediction, word):

return len(prediction) - sum(prediction[i] == word[i] for i in range(len(prediction)))

from operator import itemgetter

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

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

prediction = predict\_captcha(captcha, net)

prediction = prediction[:4]

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))

* 1. **Conclusión**

Finalmente, el software fue parcialmente exitoso, debido a que se ejecutaron correctamente los entrenamientos y la lectura de las imágenes. Pero no fue lo suficientemente exacto el de cifrado, variando sus resultados entre un 40% y 80% de éxito de las letras revisadas.

La actividad resultó ser de lo más interesante aprendimos acerca de machine learning e inteligencia artificial y aunque no fue del todo exitoso fue una buena experiencia.