ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

PLATAFORMA DE EVALUACIÓN DE ALGORITMOS DE RECONOCIMIENTO DE CARACTERES NUMÉRICOS EN IMÁGENES DIGITALES

AUTORA: JANETH ILEANA ARIAS GUADALUPE

Proyecto de investigación, presentado ante el Instituto de Posgrado y Educación Continua de la ESPOCH, como requisito parcial para la obtención del grado de

MAGISTER EN SISTEMAS DE CONTROL Y AUTOMATIZACIÓN INDUSTRIAL

RIOBAMBA - ECUADOR

FEBRERO 2017

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

CERTIFICACIÓN:

El Tribunal del PROYECTO DE INVESTIGACIÓN CERTIFICA QUE:

El trabajo de titulación titulado "PLATAFORMA DE EVALUACIÓN DE ALGORITMOS DE RECONOCIMIENTO DE CARACTERES NUMÉRICOS EN IMÁGENES DIGITALES", de responsabilidad de la Ing. Janeth Ileana Arias Guadalupe, ha sido prolijamente revisado y se autoriza su presentación.

Tribunal:

PRESIDENTE	FIRMA
Ing. Henry Ernesto Vallejo Vizhuete	
DIRECTOR	FIRMA
Ing. Alberto Arellano Aucancela	
MIEMBRO	FIRMA
Ing. Edwin Fernando Mejía Peñafiel	
MIEMBRO	FIRMA

Riobamba, Febrero 2017

DERECHOS INTELECTUALES

Yo, Janeth Ileana Arias Guadalupe, declaro que soy responsable de las ideas, doctrinas y resultados expuestos en el presente Proyecto de Investigación, y que el patrimonio intelectual generado por la misma pertenece exclusivamente a la Escuela Superior Politécnica de Chimborazo.

Janeth Ileana Arias Guadalupe
060336580-0

DEDICATORIA

El presente trabajo de investigación está dedicado a aquellas personas que son el pilar fundamental de mi vida, de manera especial a Betto.

Jane

AGRADECIMIENTO

Agradezco a papito Dios por todas sus bendiciones.

A Betto por su apoyo incondicional, tiempo y guía para culminar este trabajo investigativo.

A mis padres, abuelitos, hermanas y sobrinos.

A la Escuela Superior Politécnica de Chimborazo, Institución que contribuyó a mi formación.

A mi familia y amigos por su apoyo.

Jane

ÍNDICE DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE GRAFICAS

LISTA DE ANEXOS

RESUMEN

La investigación tuvo como objetivo evaluar las técnicas de reconocimiento de dígitos en imágenes: K-Nearest Neighbor (KNN), TESSERACT y Redes Neuronales Artificiales (RNA); lo que permitió determinar la técnica con mayor nivel de precisión que podrá ser utilizada en diferentes aplicaciones de visión artificial. Se planteó la hipótesis de investigación "La selección adecuada de la técnica de reconocimiento de caracteres numéricos en imágenes digitales permitirá determinar la técnica con el mayor grado de precisión y minimizar el consumo de recursos en computadores SBC". Se utilizó la prueba no paramétrica de Kruskal-Wallis en la comparación del grado de precisión de las técnicas con un nivel de confianza del 95%. Los indicadores que se utilizaron son: grado de precisión, tiempo empleado para el reconocimiento, cantidad de memoria ram y nivel de uso del CPU. Se utilizó una muestra ponderada de 30 fotografías tomadas a medidores de energía eléctrica de la ciudadela Las Acacias de la ciudad de Riobamba. De acuerdo a los resultados de las pruebas estadísticas se determinó que la técnica de reconocimiento con mayor grado de precisión fue KNN alcanzando un promedio de 49,33% frente a 29,833% y 22,00% de TESSERACT y RNA respectivamente. El tiempo promedio empleado por KNN para el reconocimiento fue de 1,22 segundos, en promedio se utilizó 15,7 megabytes de memoria ram y 11,64% de uso del CPU. Se utilizó OpenCV, Python y C++ bajo la distribución Raspbian de Linux para evaluar cada una de las técnicas seleccionadas. Se recomienda profundizar en el estudio de la técnica KNN enfocadas a diferentes aplicaciones de visión artificial en la industria, así como evaluar nuevas técnicas de reconocimiento de patrones como SVM, Deep Learning y Aprendizaje no Supervisado, que se han convertido en campos de investigación activos en muchas universidades del mundo

Palabras claves: <TECNOLOGÍA Y CIENCIAS DE LA INGENIERÍA>, <INGENIERÍA TECNOLOGÍA ELECTRÓNICA>, <VISION ARTIFICIAL>, <TESSERACT (ALGORITMO)>, <K-NEAREST NEIGHBOR (KNN)> <OPENCV (HERRAMIENTA)> <REDES NEURONALES ARTIFICIALES (ALGORITMO)>

ABSTRACT

The research aimed to evaluate the techniques of digit recognition in images: K-Nearest Neighbor (KNN), TESSERACT and Artificial Neural Networks (RNA); Which allowed to determine the technique with greater level of precision that can be used in different applications of artificial vision. The research hypothesis "The proper selection of numerical carácter recognition technique in digital images will allow to determine the technique with the highest degree of accuracy and to minimize the consumption of resources in SBC computers." The non-parametric Kruskal-Wallis test was used in the comparison of the degree of precision of the techniques with a confidence level of 95%. The indicators that were used are: degree of precision, time spent for recognition, amount of RAM and level of CPU usage. A sample was used of 30 photographs taken at electric power meters of Las Acacias in Riobamba city. According to the statistical tests results, was determined that the most accurate recognition technique was KNN reaching an average of 49.33% versus to 29.833% and 22.00% of TESSERACT and RNA respectively. The average time spent by KNN for the recognition was 1.22 seconds, on average 15.7 megabytes of RAM and 11.64% of CPU usage were used. OpenCV, Python and C ++ were used under the Linux Raspbian distribution to evaluate each selected techniques. It is recommended to deepen the study of the KNN technique focused on different applications of artificial visión in the industry, as well as to evaluate new recognition techniques of Pattern such as SVM, Deep Learning and Non-Supervised Leanring, which have become active research fields in many universities around the world.

Key words: / TECHNOLOGY AND ENGINEERING SCIENCES / ENGINEERING ELECTRONIC TECHNOLOGY / ARTIFICIAL VISION / TESSERACT (ALGORITHM) / K-NEAREST NEIGHBOR (KNN) / OPENCV (TOOL) / ARTIFICIAL NEURONAL NETWORKS (ALGORITHM) /.

INTRODUCCION

La visión artificial o por computadora es hoy en día una de las áreas de investigación más activas y con más desarrollo, que existen tanto el sector universitario como empresarial y tiene como objetivo fundamental proveer la capacidad humana de la visión a los sistemas electrónicos y cuyas principales aplicaciones son la detección de movimiento, el reconocimiento fácil, el reconocimiento de caracteres, entre otros.

El presente trabajo de investigación propone el desarrollo de una "PLATAFORMA DE EVALUACIÓN DE ALGORITMOS DE RECONOCIMIENTO DE CARACTERES NUMÉRICOS EN IMÁGENES DIGITALES", para lo cual se procederá a evaluar los algoritmos KNN, Tesseract y Redes Neuronales Artificiales en el reconocimiento de la lectura de consumo de energía eléctrica, en cada uno de los algoritmos se medirá: nivel de precisión, tiempo empleado en el reconocimiento, consumo de memoria ram y cpu.

Para el desarrollo de la plataforma se utilizará un sistema computacional SBC con el sistema operativo GNU/Linux, la librería OpenCV para la gestión de las imágenes digitales, el motor de reconocimiento óptico de caracteres Tesseract y los lenguajes de programación Python y C++ en los que se implementará cada uno de los algoritmos indicados.

El documento está organizado en cuatro capítulos como se indica a continuación.

El Capítulo I, corresponde a la problemática de la investigación, el planteamiento del problema, formulación y sistematización del problema, al igual que se hace referencia a la justificación, a los objetivos y la hipótesis de investigación planteada.

El Capítulo II, detalla de manera minuciosa la revisión de literatura, realizando inicialmente el estudio del arte del problema de investigación, luego se realiza una revisión amplia de los principales contenidos relacionados con las diferentes técnicas de reconocimiento de caracteres numéricos en imágenes digitales: KNN, Tesseract y Redes Neuronales Artificiales, las características de cada una de estas técnicas, así como de las herramientas software utilizadas en su implementación.

El Capítulo III, considera los métodos de investigación empleados en el presente trabajo: método científico y cuasi experimental, utilizados para la observación y recopilación de muestras de cada uno de los escenarios propuestos. Comparativo, para evaluar y comparar cada uno de los indicadores planteados e identificar el algoritmo que presente el mayor grado de precisión en el reconocimiento de los caracteres numéricos, se detalla también la metodología utilizada para la evaluación de cada uno de las técnicas seleccionadas lo que finalmente permitirá identificar el algoritmo que cumple con la hipótesis planteada

El Capítulo IV, detalla el análisis, la interpretación de resultados de las muestras tomadas, las pruebas realizadas en cada uno de los escenarios planteados con los algoritmos seleccionados, la validación de normalidad realizado a los datos obtenidos de cada uno de los indicadores y la aplicación de la prueba no paramétrica de Kruskal-Wallis para la comprobación estadística de la hipótesis de investigación planteada.

CAPÍTULO I

MARCO REFERENCIAL

En este Capítulo, se determina el enfoque u orientación general, identificación del problema, justificación, objetivos e hipótesis que se desea comprobar con el desarrollo de la investigación.

1.1. Problema de investigación

1.1.1. Planteamiento del problema

La visión artificial hoy por hoy se ha convertido en una de las áreas de la Inteligencia Artificial con mayor futuro en investigación y desarrollo, aportando con múltiples soluciones innovadoras en diferentes áreas del conocimiento como la medicina, la agricultura, la robótica, los procesos industriales, etc.

La visión asistida por computador o visión artificial es un área de las ciencias de la computación, matemáticas e ingeniería electrónica que incluye técnicas para adquirir, procesar, analizar y entender imágenes y videos del mundo real imitando la visión humana.(Pajankar, 2015)

Los campos de la visión artificial son diversos, entre los principales se encuentran:

- Control de calidad de Procesos Industriales
- Identificación e inspección de objetos
- Reconocimiento de Caracteres (OCR)
- Visión para Robótica humanoide e industrial
- Control de tráfico

El Reconocimiento de Caracteres, OCR (Optical Character Recognition), es una tecnología que engloba un conjunto de técnicas basadas en estadísticas, en las formas de los caracteres, transformadas y en comparaciones, que complementándose entre sí, se emplean para distinguir de forma automática entre los diferentes caracteres alfanuméricos existentes en una imagen digital. (Sánchez Fernández & Sadonís Consuegra, 2009).

Entre las principales aplicaciones del reconocimiento de caracteres se citan lo siguientes:

- Reconocimiento de texto manuscrito
- Reconocimiento de placas vehiculares
- Lectura automática de facturas comerciales
- Lectura automática de medidores de energía eléctrica y agua potable

1.1.2. Formulación del problema

¿La plataforma de evaluación de algoritmos de reconocimiento de caracteres numéricos en imágenes digitales mediante OpenCV, permitirá seleccionar el algoritmo más adecuado que minimice el consumo de memoria ram y procesamiento en computadores SBC de bajos costos utilizados en aplicaciones de visión artificial en la Industria?

1.1.3. Preguntas directrices

- ¿Cuáles son las aplicaciones de visión artificial de los computadores SBC en la industria?
- ¿Qué algoritmos de reconocimiento de caracteres numéricos soportadas por OpenCV fueron evaluados?
- ¿Qué nivel de precisión en el reconocimiento de dígitos se obtuvo con cada uno de los algoritmos evaluados?
- ¿Qué cantidad de memoria ram y procesamiento se consumieron en el reconocimiento de dígitos por cada uno de los algoritmos?
- ¿Cuál es el algoritmo de reconocimiento de dígitos que utilizó la menor cantidad de recursos?

1.2. Justificación de la investigación

La amplia gama de aplicaciones cubiertas por la Visión artificial, se debe fundamentalmente a que esta permite extraer y analizar información espectral, espacial y temporal de los distintos objetos incluidos en imágenes digitales.

La información espectral incluye frecuencia (color) e intensidad (tonos de gris). La Información espacial se refiere a aspectos como forma y posición (una, dos y tres Dimensiones). A su vez, la

información temporal comprende aspectos estacionarios (presencia y/o ausencia) y dependientes del tiempo (eventos, movimientos, procesos). (Sobrado, 2003)

Las principales áreas de aplicación de la visión artificial son las que a continuación se citan:

- **La medición o calibración** se refiere a la correlación cuantitativa con los datos del diseño, asegurando que las mediciones cumplan con las especificaciones establecidas. Por ejemplo, el comprobar que un cable tenga el espesor recomendado.
- **La detección de fallas** es un análisis cualitativo que involucra la detección de defectos o artefactos no deseados, con forma desconocida en una posición desconocida. Por ejemplo, encontrar defectos en la pintura de un auto nuevo, o agujeros en hojas de papel.
- **La verificación** es el chequeo cualitativo de que una operación de ensamblaje ha sido llevada a cabo correctamente. Por ejemplo, que no falte ninguna tecla en un teclado, o que no falten componentes en un circuito impreso.
- **El reconocimiento** involucra la identificación de un objeto con base en descriptores asociados con el objeto. Por ejemplo el reconocimiento de dígitos incluidos en una imagen digital.

Para las aplicaciones de reconocimiento de dígitos existen una variedad de algoritmos, el presente trabajo de investigación se centra en el desarrollo de una plataforma basada en un computador que permita estudiar y evaluar los algoritmos de aprendizaje supervisado: kNN y ARN, por su simplicidad y grado de precisión en el reconocimiento de caracteres; y contrastarlos frente a Tesseract, una aplicación opensource de reconocimiento óptico de caracteres, inicialmente desarrollado por HP y que actualmente google continua con su desarrollo.

Como es de suponerse, los recursos computacionales en los computadores de placa reducida(SBC) como la Raspberry Pi, son muy escasos, por esta razón es de fundamental importancia determinar el algoritmo de reconocimiento de dígitos más idóneo que mantenga un alto grado de precisión y un bajo consumo de los recursos computacionales de los que dispone el computador Raspberry Pi.

1.3. Objetivos

1.3.1. Objetivo general

- Evaluar y determinar el algoritmo de reconocimiento de caracteres numéricos más adecuado que minimice el consumo de recursos computacionales (memoria ram y procesamiento) en computadores SBC sin degradar el nivel de precisión en el reconocimiento mediante una plataforma opensource.

1.3.2. Objetivos específicos

- Realizar un estudio de las técnicas de pre procesamiento de imágenes y los algoritmos de reconocimiento de caracteres numéricos en imágenes digitales: KNN, TESSERAC Y ANN(Redes neuronales artificiales)
- Desarrollar una plataforma opensource para la evaluación de las técnicas.
- Determinar la relación existente entre el grado de efectividad en el reconocimiento de caracteres numéricos en imágenes digitales y los recursos hardware requeridos por cada uno de las técnicas evaluadas.
- Aplicar y verificar el adecuado funcionamiento del algoritmo seleccionado en la lectura automática de una muestra de medidores de energía eléctrica.

1.4. Hipótesis

La selección adecuada del algoritmo de reconocimiento de caracteres numéricos en imágenes digitales permitirá determinar el algoritmo con mayor grado de precisión y mínimo consumo de recursos en computadores SBC.

CAPÍTULO II

MARCO TEÓRICO

En este Capítulo, se realiza un estudio detallado de los conceptos, técnicas, herramientas y plataformas utilizadas para el desarrollo de la presente investigación, así como de investigaciones previas que motivaron el desarrollo de la misma.

2.1. Estado del arte

Varias investigaciones relacionadas con el objeto de estudio se han revisado en detalle, aportando al presente trabajo de investigación fundamentalmente para:

- Entender adecuadamente la teoría relacionada con el procesamiento digital de imágenes y el reconocimiento óptico de caracteres, así como el uso de la librería OpenCV.
- Identificar la descripción del problema y los objetivos planteados en las investigaciones realizadas.
- Conocer el proceso desarrollado por los autores para realizar las investigaciones.
- Determinar las métricas, indicadores o parámetros que los autores utilizan para realizar las pruebas.
- Determinar el aporte que realizan los autores en las investigaciones realizadas.
- Observar los resultados y conclusiones obtenidas en base a las pruebas realizadas que hacen los autores.

Las investigaciones con mayor relevancia se detallan a continuación:

Investigación: "LCD/LED DIGIT RECOGNITION BY IPHONE" (Xian Li, 2011)

La motivación del autor de la investigación nace de la necesidad de almacenar en un repositorio de datos los valores que se visualizan en las pantallas LCD y LED de los dispositivos médicos que existen en las casas, como medidores de temperatura digital, medidores de presión, etc.

La idea principal del autor es desarrollar una aplicación que le permita a un teléfono Iphone reconocer los dígitos presentes en los dispositivos y a través de la red móvil enviarlos a una base de datos segura, para su posterior análisis médico.

El proceso desarrollado contempla las siguientes etapas:

- Captura de la imagen o la toma del álbum de fotos del teléfono
- Redimensionamiento de la imagen
- Pre procesamiento para encontrar los contornos
- Reconocimiento de caracteres
- Entrega de los resultados

Para el procesamiento de la imagen se utiliza la librería OPENCV, TESSERACT es utilizado como herramienta para el reconocimiento óptico de caracteres.

Los resultados de las pruebas realizadas se muestran en la tabla 1-2.

Tabla 1-2 Resultados obtenidos de las pruebas de Reconocimiento

Row / Col	0	1	2	3	4	5	6	7	8	9
0	82. 22%	0. 00%	0. 00%	5. 26%	0. 00%	0. 00%	0. 00%	0. 00%	3. 77%	5. 77%
1	0. 00%	81. 36%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%	5. 56%	0. 00%	0. 00%
2	0. 00%	0. 00%	92. 59%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%
3	0. 00%	0. 00%	0. 00%	78. 95%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%
4	0. 00%	0. 00%	0. 00%	0. 00%	90. 57%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%
5	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%	83. 67%	4. 17%	0. 00%	0. 00%	0. 00%
6	0. 00%	0. 00%	0. 00%	0. 00%	5. 66%	10. 20%	83. 33%	0. 00%	11. 32%	0. 00%
7	0. 00%	10. 17%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%	72. 22%	0. 00%	0. 00%
8	17. 78%	0. 00%	3. 70%	8. 77%	0. 00%	0. 00%	12. 50%	0. 00%	75. 47%	9. 62%
9	0. 00%	0. 00%	0. 00%	7. 02%	0. 00%	6. 12%	0. 00%	22. 22%	9. 43%	84. 62%
No result	0. 00%	8. 47%	3. 70%	0. 00%	3. 77%	0. 00%	0. 00%	0. 00%	0. 00%	0. 00%
Total counts	45	59	54	51	53	49	48	36	53	52

Fuente: Xian Li, 2011

Luego de realizar las pruebas se concluye que la aplicación desarrollada tiene un nivel adecuado de precisión en el reconocimiento de los dígitos, tanto el algoritmo para encontrar los contornos y la herramienta Tesseract no consumen desmesuradamente los recursos del teléfono.

Sin embargo en este trabajo de investigación no se prueban otras alternativas para el reconocimiento así como no hay resultados de los recursos de procesamiento y memoria utilizados.

Investigación: "RECONOCIMIENTO OPTICO DE CARACTERES EN IMÁGENES DIGITALES DE CONTADORES DE GAS" (Martín de Loaches, Fernandez, 2015)

El trabajo de investigación pretende resolver la necesidad que existe en la empresa Red de Gas de la ciudad de Madrid en España, para lo cual los autores desarrollan una aplicación en JAVA que permite obtener el número de referencia del autor así como el valor del consumo.

El proceso que se sigue para el trabajo de investigación es el siguiente:

- Carga de las imágenes
- Transformación de la imagen a escala de grises
- Localización de las zonas de interés
- Binarización
- Lectura de los caracteres específicos

Se realizan pruebas para cada una de las etapas, sin embargo los autores no muestran en detalle los resultados obtenidos, en el que se verifique el nivel de precisión del reconocimiento de los dígitos.

En este trabajo de investigación no se utiliza OPENCV y no hay pruebas del consumo de recursos utilizados.

Investigación: "DISEÑO E IMPLEMENTACIÓN DE UN MÓDULO DE RECONOCIMIENTO DE NÚMEROS MANUSCRITOS" (Garrido, 2010)

El autor describe las diferentes aplicaciones que tiene el reconocimiento de caracteres manuscritos como la lectura de facturas, cheques, papeletas de votación, encuestas, etc. Para esto implementa un módulo en MATLAB mediante redes neuronales.

El proceso desarrollado es el siguiente:

- Segmentación de la imagen
- Normalización
- Clasificación de la imagen
- Reconocimiento de caracteres

Se desarrollan pruebas de reconocimiento con una muestra de 150 secuencias de números escritos en una hoja cuadriculada. El detalle de los resultados se muestra en la tabla 2-2:

Tabla 2-2 Resultados de reconocimiento de números manuscritos

Números	**Cantidad de Patrones**	**% Reconocidos**	**% Eficiencia**
0	200	99.00	98.50
1	200	96.50	96.50
2	200	96.50	95.50
3	200	92.00	91.00
4	200	99.50	99.50
5	200	96.50	95.50

6	200	97.50	96.00
7	200	96.00	95.50
8	200	96.00	94.50
9	200	97.50	96.50

Fuente: Garrido,2010

Finalmente, el autor concluye que el reconocimiento de las muestras,obteniendo una presición del 95,9%, el 3,3% de las muestras no fueron reconocidas, mientras que el 0,8% tuvo errores en el reconocimiento.

Investigación: "CONVERSION DE TEXTO MANUSCRITO A FORMATO DIGITAL UTILIZANDO MÁQUINAS DE SOPORTE VECTORIAL" (Cisneros, 2007)

El problema que intenta resolver el autor es reconocer de forma automática caracteres manuscritos a partir de un documento previamente digitalizado. Para ello se implementa un algrotimo basado en Máquinas de Soporte Vectorial en Matlab.

La metodología utilizada para resolver el problema se divide en tres etapas:

- Preprocesameniento de la Imagen: Que contempla el umbralado, detección de bordes, dilatación, relleno y segmentación de la imagen.
- Extracción de características
- Clasificación
- Reconocimiento

Para las pruebas se utilizarón 200 muestras de texto en letras minúsculas, mayúsculas y números, comparándo los resultados obtenidos con los algoritmos basados en Máquinas de soporte vectorial, Redes Neuronales Artificiales y el clasificador euclidiano, la tabla 3-2, 4-2 y 5-2 muestra los resultados obtenidos.

Tabla 3-2 Resultados del reconocimiento con letras minúsculas

Letras	MSV	ANN	Euclidiano
a	100	100	90
b	100	100	90
c	90	80	70
d	100	100	90
e	100	100	80
f	80	80	80
g	90	80	70
h	90	90	80
i	100	100	100
j	80	90	80
k	80	80	70
l	100	90	90
m	100	90	80
n	80	90	80
o	100	100	80
p	100	100	90
q	90	100	90
r	100	100	100
s	80	80	70
t	80	80	60
u	80	70	60
v	80	70	70
w	90	80	80
x	90	100	80
y	80	80	70
z	80	80	70
	90%	89.1%	80%

Fuente: Cisneros, 2007

Tabla 4-2 Resultados del reconocimiento con números

NUMEROS	MSV	ANN	EUCLIDIANO
0	100%	100%	90%
1	100%	100%	90%
2	90%	90%	90%
3	70%	70%	60%
4	90%	90%	80%
5	60%	50%	50%
6	100%	100%	90%
7	100%	100%	80%
8	100%	100%	90%
9	100%	100%	90%
Total	90.10%	90%	81%

Fuente: Cisneros, 2007

Tabla 5-2 Resultados del reconocimiento con letras mayúsculas

Letras	MSV	ANN	Euclidiano
A	90	90	80
B	100	100	90
C	80	80	70
D	100	100	90
E	100	100	80
F	80	80	80
G	80	80	70
H	90	90	80
I	100	100	100
J	80	90	80
K	80	80	70
L	100	90	90
M	100	90	80
N	80	90	80
O	100	100	80
P	100	100	90
Q	90	90	80
R	100	100	100
S	80	80	70
T	70	80	70
U	80	70	60
V	80	70	70
W	100	80	80
X	90	100	80
Y	80	80	70
Z	80	80	70
	88.8461538	**88,07 %**	**79.20%**

Fuente: Cisneros, 2007

El autor concluye que el algoritmo basado en máquinas de soporte vectorial tiene una precisión de reconocimiento del 89,65% frente al 89,06 y 80,01 de los algoritmos basados en redes neuronales y clasificadores euclidianos respectivamente.

2.2. Visión por Computador

2.2.1. Introducción

La visión por computador puede considerarse como un subcampo dentro de la inteligencia artificial y se centra básicamente en intentar expresar el proceso de visión en términos de computación. Algunas de las técnicas más utilizadas en el campo de la visión por computador son el procesamiento de imágenes, los gráficos por computador y el reconocimiento de patrones. (Pajares, 2008)

Es importante destacar que el principal objetivo a alcanzar por todas las investigaciones realizadas en este campo, es intentar conseguir la máxima similitud entre la visión artificial realizada por los computadores y la visión biológica utilizada por los seres vivos.

La visión es, sin duda, nuestro sentido más poderoso y a la vez el más complejo, esta quizás es la razón fundamental por la que hasta el momento no existe un sistema de visión por computador que sea 100% idéntico al humano, lo que provoca que más investigaciones sobre el tema se realicen diariamente.

La importancia de todas las investigaciones realizadas en el campo de la visión por computador se hace evidente al observar la alta demanda por parte de la industria y la sociedad de aplicaciones capacitadas para reconocer objetos. Como ejemplo de este tipo de aplicaciones destacaríamos el caso de Fujitsu que ha diseñado robots que tienen la capacidad de reconocer imágenes en tres dimensiones. También es importante destacar el uso de estas tecnologías aplicadas en los nuevos buscadores de internet. Proyectos como Riya ó VIMA entre otros investigan la creación de buscadores de páginas web que además de poder localizar texto como puede hacer google, permitan buscar contenidos multimedia gracias al uso de algoritmos de reconocimiento de imágenes. (Sanz, 2008)

Los objetivos típicos de la visión por computadora incluyen los siguientes:

- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes (por ejemplo, caras humanas, dígitos en una factura digital, etc.)
- Registro de diferentes imágenes en una escena u objeto.
- Seguimiento de un objeto en una secuencia de imágenes.
- Mapeo de una escena para generar un modelo tridimensional de la misma.
- Estimación de las posturas tridimensionales de los humanos.
- Búsquedas de imágenes digitales por su contenido.

Estos objetivos se consiguen por medio del reconocimiento de patrones, aprendizaje estadístico, geometría de proyección, procesamiento de imágenes, teoría de grafos y otros campos de la ciencia.(Ingelmo, 2009)

2.2.2. Arquitectura de un Sistema de visión por computador

La visión por computador lleva asociada una enorme cantidad de conceptos relacionados con hardware, software y también con desarrollos teóricos. En esta sección se verán los pasos fundamentales para llevar a cabo una tarea de visión artificial.

Para cumplir satisfactoriamente su tarea, todo sistema de reconocimiento de caracteres debe realizar cuatro fases, como se muestra en la Figura 1-2.

Figura 1-2 Fases del reconocimiento de caracteres
Realizado por: Arias Janeth, 2016

Cada una de las fases indicadas en la Figura 1-2, contemplan una variedad de técnicas y métodos los cuales seleccionados adecuadamente, permiten tener un grado de precisión alto en el reconocimiento de caracteres o dígitos en imágenes digitales, considerando además el mínimo consumo de recursos de los sistemas computacionales involucrados en el proceso de reconocimiento.

Las técnicas de pre procesado pretenden mejorar o realzar las propiedades de las imágenes para facilitar las siguientes operaciones de la Visión Artificial, tales como las etapas de segmentación, extracción de las características y finalmente la interpretación automática de las imágenes. Estás técnicas básicamente son: Realce o aumento del contraste, suavizado o eliminación del ruido presente en la imagen y la detección de bordes.(Dueñas, 2014)

Generalmente el preprocesado pretende reparar en la imagen los desperfectos producidos o no eliminados por el hardware: deformación de ésta, ruido introducido, poco o mucho contraste o

brillo, falta de ecualización apropiada, etc. Los algoritmos de preprocesado permiten modificar la imagen para eliminar ruido, transformarla geométricamente, mejorar la intensidad o el contraste, etc.; procesos que intentan mejorar el resultado final de la imagen capturada. Por supuesto, como estos algoritmos necesitan gran cantidad de tiempo de procesado, es lógico que lo mejor es utilizar un hardware conveniente que los evite.

En un proceso de visión artificial estos algoritmos tienen que ser utilizados lo menos posible, un uso excesivo de ellos repercutirá en el tiempo de proceso total e indicará que la calibración, iluminación y selección de los elementos de la etapa de adquisición no ha sido la adecuada.

A parte de la degradación de la imagen, existen características de la imagen que, en muchos casos, es conveniente mejorar. Así, muchas veces, se requiere mejorar el contraste, brillo, niveles de grises, eliminar brillos, aumentar los bordes, mejorar las texturas, etc.

De lo anteriormente expuesto podemos concluir que el objetivo principal del pre-procesamiento es mejorar la imagen de forma que el objetivo final de la tarea tenga mayores posibilidades de éxito.

La fase de segmentación tiene como objetivo la detección de los contornos de los caracteres o símbolos de la imagen tomando en cuenta la información de intensidad de los mismos. Además permite la descomposición del texto en diferentes entidades lógicas suficientemente invariantes para no depender del estilo de la escritura y que puedan ser suficientemente significativas para reconocerlas. Los principales métodos utilizados para esta fase son: método de agrupamiento, método basado en histogramas, método de crecimiento de regiones, método de particionamiento gráfico entre otros. (Sánchez Fernández & Sadonís Consuegra, 2009)

Una vez realizada la segmentación, se tiene una imagen normalizada en la que se encuentra la información susceptible de ser “reconocida”. La información así representada, una matriz bidimensional de valores binarios, niveles de gris o color RGB, no codifica de forma óptima las características más discriminativas del objeto al que representa. La extracción de las características es una de las fases más difíciles en los sistemas de reconocimiento de caracteres, puesto que es muy difícil escoger un conjunto de características óptimo. Para esta fase se utilizan entre otros los siguientes métodos: análisis de componentes principales (PCA), análisis discriminante lineal (LDA), análisis independiente de componentes (ICA) y análisis discriminante no lineal (NDA). (Sánchez Fernández & Sadonís Consuegra, 2009)

Una vez que se obtienen las características más importantes de cada uno de los caracteres incluidos en la imagen digital, el siguiente paso es realizar el reconocimiento del mismo en base a un patrón predefinido, básicamente existen dos métodos de reconocimiento: offline y online.

Los métodos offline más utilizados son: Clustering, Feature Extraction, Pattern Matching y Redes Neuronales, mientras que el clasificador k-NN y el algoritmo basado en dirección son los métodos online más comunes. (Dedgaonkar, Chandavale, & Sapkal, 2012)

2.2.3. Aplicaciones

El número de aplicaciones relacionadas con la Visión Artificial aumenta cada día. En la tabla 6-2 adjunta se citan algunos de los campos donde es empleada esta disciplina.

Tabla 6-2 Aplicaciones de la visión por computador

ÁREA DE PRODUCCIÓN	APLICACIONES
Control de calidad	• Inspección de productos (papel, aluminio, acero, etc.). • Identificación de piezas. • Etiquetados (fechas de caducidad). • Inspección de circuitos impresos. • Control de calidad de los alimentos (frutas, embutidos, etc.).
Robótica	• Control de soldaduras. • Guiado de robots (vehículos no tripulados).
Biomédica	• Análisis de imágenes de microscopía (virus, células, proteínas). • Resonancias magnéticas, tomografías, genoma humano.
Reconocimiento de caracteres	• Control de cheques, inspección de textos, etc.
Control de tráfico	• Matrículas de vehículos. • Tráfico vial.
Agricultura	• Interpretación de fotografías aéreas. • Control de plantaciones.

Realizado por: Arias Janeth, 2016
Fuente:(Aplicaciones de la Visión Artificial, 1993)

2.3. Técnicas de reconocimientos de imágenes

En esta sección se describen las técnicas más importantes para el reconocimiento óptico de caracteres. De cada una de las técnicas vistas se detallan sus fundamentos y se señalan sus características más importantes.

2.3.1. Clasificador KNN

Una tarea muy importante en el reconocimiento de patrones es la clasificación, tarea que sirve para distinguir objetos en una imagen, este reconocimiento automático se lo realiza mediante la transformación de este objeto a un vector llamado patrón, características determinantes o rasgos permiten discriminar a que clase corresponde un objeto dentro de la imagen.(Serpa Andrade, 2014)

El algoritmo KNN (K vecinos más cercanos) es un método de clasificación supervisada que sirve para estimar la probabilidad de que un elemento x pertenezca a la clase C (j) a partir de la información proporcionada por el conjunto de prototipos, donde k determina el número de vecinos que son considerados para realizar la clasificación.(Ingelmo, 2009)

En el reconocimiento de patrones, el algoritmo kNN es usado como método de clasificación de objetos basados en un entrenamiento mediantes ejemplos cercanos en el espacio de los elementos.

Este clasificador es una extensión de la regla de clasificación NN, conocido como la regla del vecino más cercano (Nearest Neighbour). La idea fundamental en la que se basa la regla del vecino más cercano reside en considerar que las muestras pertenecientes a una misma clase se encuentran probablemente próximas en el espacio de representación. Para determinar la cercanía entre muestras, la regla utiliza el concepto métrica de distancia, donde existen diferentes maneras de realizar esta medición. Por lo tanto, debido al conjunto de N muestras pasadas con clases definidas y a causa de una muestra nueva x de la que se desconoce su clase, esta se clasificará o etiquetará con la clase en la que la medida de la distancia de la nueva muestra x, con un elemento de la clase seleccionada, presentó el valor mínimo. Este procedimiento no aprovecha toda la información que se puede extraer del conjunto de entrenamiento, por tal motivo se crea la regla K-NN, la que además de encontrar el vecino más cercano a la nueva muestra, tiene en cuenta el entorno que rodea a la misma contando el número de muestras (K) que se encuentran próximas. Así, si se tiene un conjunto de muestras para cada clase, la clasificación se basa en las K muestras más próximas a la nueva muestra. (Montoya Holguín, Cortés Osorio, & Chaves Osorio, 2014)

Por ejemplo como se observa en la figura 2-2, la primera vecindad encerrada con el círculo más pequeño (k=5) indica que la nueva muestra (forma de estrella) será clasificado como círculo, debido a que en esta región hay tres círculos contra dos rombos, pero si se selecciona una región mayor como el círculo punteado más alejada de la estrella (k=9), la estrella será clasificada como rombo, pues en esta región hay cinco rombos contra cuatro círculos.

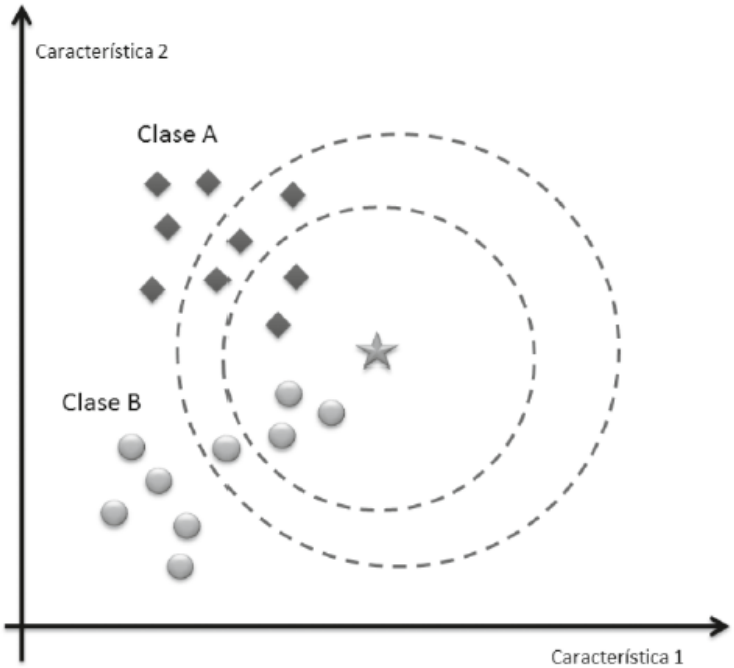


Figura 2-2 Ejemplo de clasificación k-NN
Fuente: (Serpa Andrade, 2014)

2.3.2 Tesseract

Tesseract es una biblioteca de código abierto para reconocimiento óptico de caracteres, que fue originalmente desarrollado por Hewlett Packard.

Tesseract es probablemente el motor de OCR de código abierto más preciso del que actualmente se dispone. En combinación con la biblioteca de procesamiento de imágenes Leptonica se puede leer una amplia variedad de formatos de imagen y convertirlos en texto en más de 60 idiomas. Fue uno de los 3 primeros motores en la prueba de precisión que en 1995 realizó la Universidad de Las Vegas. Entre 1995 y 2006 se había hecho muy poco trabajo en él, pero desde entonces se ha mejorado ampliamente por Google. Actualmente está liberada bajo la Licencia Apache 2.0. (Henno, 2015)

2.3.3. Redes Neuronales Artificiales

Las redes neuronales artificiales tratan de emular el funcionamiento del cerebro humano, creando conexiones entre unidades de procesamiento denominadas neuronas. La unidad central de la red es una neurona artificial que ha sido modelada matemáticamente a partir de una neurona biológica.

Las dendritas y la sinapsis proveen el medio para conectar la neurona con sus vecinas, se ha encontrado que una neurona puede estar conectada con una gran cantidad de neuronas, de capas diferentes y que además de intercambiar información intercambian otro tipo de proteínas. En el núcleo se procesa la información recibida y se prepara para propagar o inhibir la información analizada mediante el axón.(Medina Sanchez, 2015)

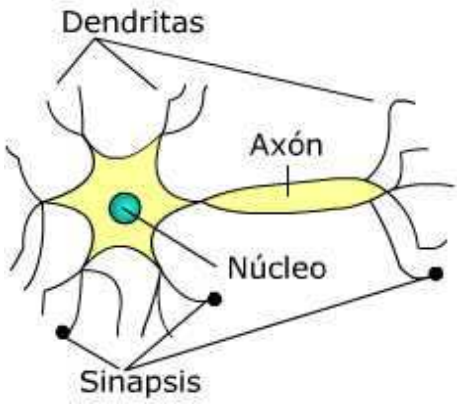


Figura 3-2 Neurona biológica
Fuente:http://www.virtual.unal.edu.co/cursos/ingenieria/2001832/lecciones/ cap_4/images/neurona.jpg

El aprendizaje biológico consiste en fortalecer las uniones de las neuronas involucradas en el proceso. Cuantas más neuronas sean involucradas mejor serán los resultados.

Por otra parte la neurona artificial cuenta con similares características aunque a un nivel limitado de información y número de conexiones. La neurona posee entradas con sus respectivos pesos, los mismos que simularán el proceso de fortalecimiento de los enlaces mediante un valor numérico. Éste número puede ser negativo si se trata de inhibir la entrada.

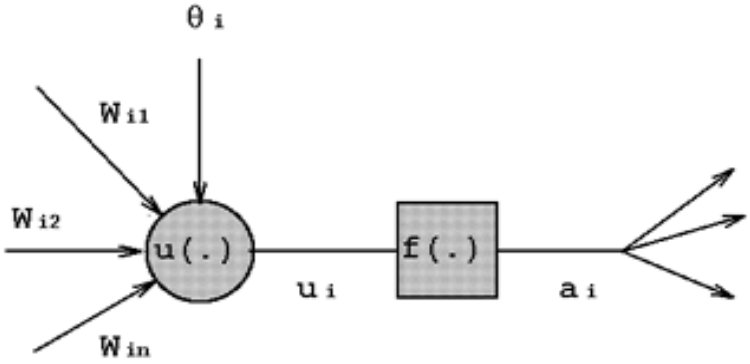


Figura 4-2 Neurona artificial
Fuente: **http:**//www.virtual.unal.edu.co/cursos/ingenieria/2001832/lecciones/c ap_4/images/escalon.gif

Es conveniente acotar las salidas de la neurona mediante funciones de transferencia como la tangente hiperbólica y sigmoidal.(Medina Sanchez, 2015)

El esfuerzo por simular la inteligencia humana radica en varias ventajas como la solución de problemas no lineales, la distribución de la información en cada una de las neuronas, de manera que si una parte de la red es afectada, el rendimiento se ve disminuido pero el sistema no colapsa totalmente. Otra característica a destacar es que la misma estructura de red, se puede utilizar para solucionar diversos problemas, sin necesidad de cambiar el código fuente.

La desventaja de las redes es que necesitan de una gran cantidad de muestras para su entrenamiento, también la necesidad de varias horas para el proceso de aprendizaje.

Debido a las características que presenta la red neuronal la gama de aplicaciones es muy grande y abarca temas como el procesamiento de señales, filtros de ruido, procesamiento de lenguaje, reconocimiento de patrones, entre otros.(Medina Sanchez, 2015)

2.3.4. Arquitectura de las Redes Neuronales Artificiales

La topología o arquitectura de las RNA hace referencia a la organización y disposición de las neuronas de la red formando capas de procesadores interconectados entre sí a través de sinapsis unidireccionales. (Flores López & Fernández Fernández , 2008)

Las redes neuronales se pueden clasificar en según los criterios que a continuación se citan:

1. Por su estructura en Capas
2. Por el Flujo de datos en la RNA
3. Por el tipo de respuesta de la RNA

Según la estructura de las capas una RNA puede ser:

1. Redes Monocapa, compuesta por una única capa de neuronas entre las que se establecen conexiones laterales y en ocasiones, autorecurrentes. Este tipo de RNA se suele utilizar en la resolución de problemas de autoasociación y clusterización. Una de las redes más representativas de este modelo es la red de Hopfield, que ha tenido una gran influencia en el desarrollo posterior de redes neuronales.
2. Redes Multicapa, en este tipo de RNA, las neuronas se organizan en varias capas (entrada, ocultas y de salida)

Las RNA por el flujo de datos pueden ser:

1. Redes unidireccionales o de propagación hacía adelante, en este tipo de redes ninguna salida neuronal es entrada a unidades de la misma capa o de capas precedentes. La información circula en único sentido.

2. Redes de propagación hacia atrás, estas redes se caracterizan por que las salidas de las neuronas pueden servir de entrada a unidades del mismo nivel o de niveles previos.

Finalmente, las RNA por el tipo de respuesta pueden ser:

1. Redes heteroasociativas, entrenadas para que ante la presencia de un patrón A, el sistema responda con un patrón B. Estas redes precisan al menos de dos capas, una para captar y retener la información de entrada y otra para mantener la salida con la información asociada, generalmente se utilizan en clasificación y asociación de patrones.
2. Redes autoasociativas, entrenadas para que asocien patrones consigo misma, este tipo de redes pueden implementarse con una única capa, utilizadas normalmente en tareas de filtrado de información para analizar las relaciones de vecindad entre los datos considerados y para resolver problemas de optimización.

La tabla 7.2 resume los modelos más conocidos de RNA Heteroasociativas y Autoasociativas.

Tabla 7-2 Clasificación de las Redes Neuronales Artificiales

RNA Heteroasociativas	**RNA Autoasociativas**
• Perceptrón • Adaline • Madaline • Backpropagation • Linear Reward Penalty • Associative Reward Penalty • Adaptative Heuristic Critic • Boltzmann Machine • Cauchy Machine • Learning Matrix • Temporal Associative Memory • Linear Associative Memory • Optimal Linear Associative Memory • Drive-Reinforcement • Fuzzy Associative • Counterpropagation • Bidirectional Associatibe Memory (BAM) • Adaptive BAM	• Brain-State-in-a-box • Hopfield • Additive Grossberg • Shuting Grossberg • Self Organising Feature Map (SOFM)

• Cognitron • Neocognitron • Learning Vector Quantizer	

Realizado por: Arias Janeth, 2016
Fuente:(Aplicaciones de la Visión Artificial, 1993)

2.3.5. Tipos de Entrenamiento de las Redes Neuronales Artificiales

Durante la operación de una red neuronal se distingue claramente dos fases o modos de operación: la fase de aprendizaje o entrenamiento, y la fase de operación o ejecución. (Bertona, 2005)

Durante la primera fase, la fase de aprendizaje, la red es entrenada para realizar un determinado tipo de procesamiento. Una vez alcanzado un nivel de entrenamiento adecuado, se pasa a la fase de operación, donde la red es utilizada para llevar a cabo la tarea para la cual fue entrenada.

Una vez seleccionada el tipo de neurona artificial que se utilizará en una red neuronal y determinada su topología es necesario entrenarla para que la red pueda ser utilizada.

Partiendo de un conjunto de pesos sinápticos aleatorio, el proceso de aprendizaje busca un conjunto de pesos que permitan a la red desarrollar correctamente una determinada tarea. Durante el proceso de aprendizaje se va refinando iterativamente la solución hasta alcanzar un nivel de operación suficientemente bueno.

El proceso de aprendizaje se puede dividir en tres grandes grupos de acuerdo a sus características.

- Aprendizaje supervisado. Se presenta a la red un conjunto de patrones de entrada junto con la salida esperada. Los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada.

- Aprendizaje no supervisado. Se presenta a la red un conjunto de patrones de entrada. No hay información disponible sobre la salida esperada. El proceso de entrenamiento en este caso deberá ajustar sus pesos en base a la correlación existente entre los datos de entrada.

- Aprendizaje por refuerzo. Este tipo de aprendizaje se ubica entre los dos anteriores. Se le presenta a la red un conjunto de patrones de entrada y se le indica a la red si la salida obtenida es o no correcta. Sin embargo, no se le proporciona el valor de la salida esperada. Este tipo de aprendizaje es muy útil en aquellos casos en que se desconoce cuál es la salida exacta que debe proporcionar la red.

2.4. Software de análisis y procesamiento de imágenes

Debido a la necesidad de disponer de un paquete de visión por computador y procesamiento de imágenes lo suficientemente flexible como para proporcionar herramientas de alto nivel para el desarrollo de aplicaciones de visión artificial y la potencia, escalabilidad y disponibilidad del código fuente para el desarrollo de las aplicaciones, propicia una búsqueda incesante de software que cumpliera dichas características y sobre todo que fuese gratuito, estuviese soportado/revisado por personal cualificado y además se le intuyese una larga vida. Son muchos los paquetes de procesamiento de imágenes comerciales y software libre disponibles actualmente, y muchas las ventajas e inconvenientes de cada uno de ellos. Entre los distintos paquetes comerciales disponibles actualmente destacan por su potencia Khoros, eVision, HIPS, Exbem, Aphelion, Vision the National Instrument (IMAQ), y por supuesto paquetes de propósito general con ciertas funciones de aplicación como es el caso The Matrox Imaging Libraries (MIL) o el Toolboox Image de MatLab sin embargo, el principal inconveniente es su elevado precio y su ciclo de actualización, muchas veces, relativamente largo. Algunos de ellos carecen de un entorno de desarrollo de alto nivel (i.e. HIPS), otros disponen de éste, pero están ligados a la plataforma de desarrollo (i.e. Khoros - Unix, Linux; Exbem - MacOS; eVision, Aphelion - Windows) o al propio hardware de captura (i.e. MIL). Todos ellos proporcionan funciones de procesamiento y análisis de imágenes, reconocimiento de patrones, estadísticas, calibración de la cámara, etc. a través del propio entorno o a través de librerías de funciones, desarrollados en la mayoría de las ocasiones en C/C++. Sin embargo, tan sólo HIPS pone a disposición del cliente su código fuente, y en la mayoría de los casos hablamos de librerías monolíticas, muy pesadas y no demasiado rápidas. Por otro lado, son muchos los paquetes no comerciales (con y sin licencia Software Libre) disponibles en el mercado, entre ellos destacan OpenCV, Gandalf, TargetJr, VXL (basado en TargetJr), CVIPTools, ImageLib, ImLib3D (Linux), LookingGlass, NeatVision (Java), TINA y XMegaWave (Unix/Linux). Todos ellos disponen de herramientas para el procesamiento de imágenes, pero a excepción de OpenCV y Gandalf ninguno proporciona un marco de trabajo completo para el desarrollo de aplicaciones relacionadas con la visión por computador. Esta capacidad de desarrollo contempla, no sólo el procesamiento de imágenes, sino tareas mucho más complejas como el reconocimiento de gestos, estimación del movimiento y la posición de un objeto, morphing, estimadores (filtros de Kalman, etc.), etc. Sin embargo, sólo OpenCV proporciona bibliotecas de tipos de datos estáticos y dinámicos (matrices, grafos, árboles, etc.), herramientas con la posibilidad de trabajar con la mayoría de las capturadoras/cámaras del mercado, entornos de desarrollo fáciles

e intuitivos y todo ello, corriendo en dos de los sistemas operativos más utilizados del mundo Microsoft® Windows y Linux. (Arévalo, González, & Ambrosio, 2002)

A continuación se detalla la estructura y las principales características de la librería OpenCV.

El 13 de Junio del 2000, Intel® Corporation anunció que estaba trabajando con un grupo de reconocidos investigadores en visión por computador para realizar una nueva librería de estructuras/funciones en lenguaje C. Esta librería proporciona un marco de trabajo de nivel medio-alto que ayuda a desarrollar nuevas formas de interactuar con los computadores. Este anuncio tuvo lugar en la apertura del IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). Este acontecimiento marcó el nacimiento The Open Computer Vision Library y lo hacía bajo licencia BSD (Software Libre). La librería OpenCV es una API de aproximadamente 300 funciones escritas en lenguaje C que se caracterizan por lo siguiente:

- Su uso es libre tanto para su uso comercial como no comercial.
- No utiliza librerías numéricas externas, aunque puede hacer uso de alguna de ellas, si están disponibles, en tiempo de ejecución.
- Es compatible con The Intel® Processing Library (IPL) y utiliza The Intel® Integrated Performance Primitives (IPP) para mejorar su rendimiento, si están disponibles en el sistema, entre otras. (Arévalo et al., 2002)

Estructura y características de la librería OpenCV

La librería OpenCV está dirigida fundamentalmente a la visión por computador en tiempo real. Entre las diversas aplicaciones se destacan las siguientes:

- Interacción hombre-máquina.
- Segmentación y reconocimiento de objetos.
- Reconocimiento de gestos.
- Seguimiento del movimiento.
- Estructura del movimiento en robots, etc.

Como podemos ver en la figura 5-2, la librería OpenCV proporciona numerosos elementos de alto nivel que facilitan sobre manera el trabajo al usuario. Por ejemplo, proporciona filtros Microsoft® DirectShow para realizar tareas tales como: calibración de la cámara, seguidores de objetos (Kalman tracker y ConDensation tracker), etc. Todos ellos se pueden utilizar en Microsoft® GraphEdit. (Arévalo et al., 2002)

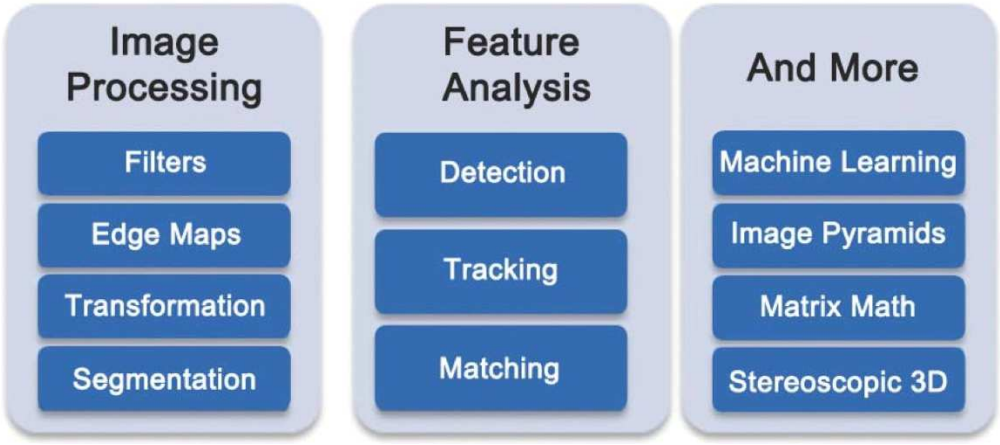


Figura 5-2 Estructura de la librería openCV
Fuente: (Arévalo et al., 2002)

2.5. Computador SBC Raspberry Pi

Por otro lado el aparecimiento de computadores SBC como la Raspberry Pi han despertado un gran interés en los investigadores que están desarrollando nuevas e innovadoras aplicaciones en diferentes ámbitos. Raspberry Pi es un computador de placa reducida del tamaño de una tarjeta de crédito desarrollado en el reino unido por la Raspberry Pi Foundation, con el objetivo de estimular la enseñanza de las ciencias de computación básicas en las escuelas. (Sarthak Jain, Anant Vaibhav, 2014)
Existen dos modelos cuyas principales características se detallan a continuación en la tabla 7-2:

Tabla 8-2 Características de los Modelos del Computador RASPBERRY PI

Recurso	Modelo A	Modelo B
SoC	Broadcom BCM2835	Broadcom BCM2836
Memoria RAM	256 Mb	512 Mb
Puertos USB	1	2
Puerto Ethernet	Ninguno	10/100 Mbps
Potencia requerida de alimentación	300 mA(1.5 mW)	700 mA (3.5 mW)

Realizado por: Arias Janeth, 2016
Fuente: (Warren Gay, 2014)

De igual manera INTEL ha realizado un gran aporte a la comunidad científica al poner a disposición OpenCV: un conjunto de librerías para la manipulación de imágenes digitales.

OpenCV (Open Source Computer Vision) es un conjunto de librerías de código abierto que contiene más de 300 algoritmos optimizados para análisis de imágenes y video. Originalmente

desarrollado por INTEL, en el grupo de investigación liderado por Gary Bradski. (Robert Laganiere., 2014)

Las principales aplicaciones de OpenCV son: (Yu, Cheng, Cheng, & Zhou, 2004)

- Interacción Hombre-Máquina
- Segmentación y reconocimiento de objetos
- Análisis y procesamiento de imágenes
- Reconstrucción 3D
- Reconocimiento de gestos
- Seguimiento de objetos
- Reconocimiento de caracteres, entre otros.

2.6 Selección de la cámara

Aunque existe la posibilidad de conectar diferentes tipos de cámaras web mediante los puertos USB de la tarjeta SBC Raspberry pi, el fabricante recomienda utilizar el módulo de cámara modelo RPI-CAM-V2_1 (figura 6-2) que se conecta directamente al conector CSI de 15 pines disponible en la tarjeta.

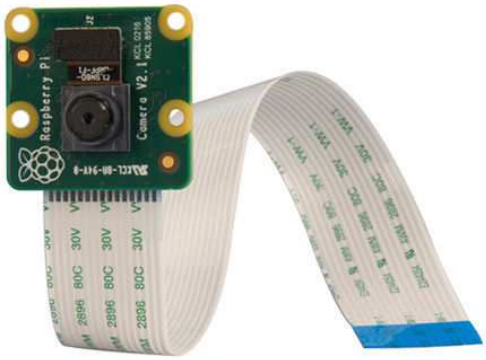


Figura 6-2 Módulo de cámara para Raspberry Pi
Fuente: http://www.raspberry.org

La cámara cuenta con un sensor sony IMX219 de 8 megapíxeles que además de capturar video en alta definición en los formatos: 1080p30, 720p60 y 640x480p90, es capaz de tomar fotos con una resolución de 3280 x 2464 píxeles.

La cámara puede ser manipulada directamente mediante scripts desarrollados por la comunidad Python, que se instalan como se muestra en la figura 3.

```
/home/tesis/$ sudo apt-get update
/home/tesis/$ sudo apt-get upgrade
/home/tesis/$ sudo apt-get install python-picamera
```

Figura 7-2 Instalación de software de gestión de la Picamera
Realizado por: Arias Janeth, 2016

CAPITULO III

METODOLOGÍA DE LA INVESTIGACIÓN

El presente capítulo detalla los principales aspectos metodológicos que se llevaron a cabo para cumplir con los objetivos propuestos en el trabajo de investigación.

3.1 Diseño de la investigación

El diseño de la presente investigación es del tipo cuasi experimental ya que se escoge la técnica de reconcomiendo de dígitos más adecuada en imágenes para lo cual se evaluarán: KNN, Tesseract y Redes Neuronales Artificiales en fotografías tomadas a medidores de energía eléctrica con el objetivo de obtener la lectura de consumo de manera automática, además los datos de las pruebas realizadas son generados por el autor de esta investigación.

De acuerdo a Hernández (2003), los diseños cuasi experimentales manipulan deliberadamente al menos una variable independiente para observar su efecto y relación con una o más variables dependientes. En el caso específico de la presente investigación se determinará el porcentaje de precisión en el reconocimiento de los dígitos así como el tiempo, el consumo de memoria ram y la cantidad de procesamiento utilizados para esta tarea. (Hernández, 2010)

3.2 Tipo de investigación

El presente trabajo de investigación puede clasificarse en aplicativa y experimental.

- **Aplicativa:** ya que se basa en conocimientos existentes, derivados de investigaciones previas, dirigida al desarrollo tecnológico para establecer nuevos procesos para mejorar los existentes.
- **Experimental:** ya que se basa en pruebas realizadas en escenarios de laboratorio, en las que se observa los elementos más importantes del objeto de estudio que se investiga para obtener una captación de los fenómenos a primera vista.

3.3 Métodos y técnicas

Los métodos y técnicas utilizados para la presente investigación son:

3.3.1. Métodos

- Para la presente investigación se utilizará el Método de Análisis que permitirá la identificación de cada una de las partes que caracterizan la realidad del objeto de estudio para de esta manera establecer la relación causa-efecto entre los elementos que componen el mismo.
- También se utilizará el Método Científico que es el proceso mediante el cual una teoría científica es validada o bien descartada y contiene una serie ordenada de pasos a seguir para la resolución de un problema determinado y cuyas etapas son: observación, planteo de un problema, recopilación de datos, formulación de hipótesis, experimentación y conclusión.

3.3.2. Técnicas

Las técnicas que serán utilizadas en la presente investigación son:

- **Búsqueda de información:** permite obtener la información necesaria acerca del objeto de estudio de la investigación para su desarrollo, utilizando las fuentes secundarias disponibles.
- **Pruebas:** permite realizar experimentos en escenarios de laboratorio.
- **Observación:** permite determinar resultados de las pruebas realizadas en los escenarios de laboratorio.
- **Análisis:** permite determinar los resultados de la investigación

3.4. Instrumentos

Los instrumentos para recopilar los datos de los indicadores son los siguientes:

- **OpenCV:** librería de software libre para aplicaciones de visión artificial, dispone de interfaces de programación para C, C++, Python y JAVA. (Comunidad OPENCV: http://www.opencv.org). La comunidad de OpenCV cuenta actualmente con 47000 usuarios y más de 9 millones de descarga lo que le convierte en la plataforma número uno de investigación y desarrollo en el campo de la visión artificial.
- **Tesseract:** motor libre para aplicaciones de reconocimiento óptico de caracteres actualmente desarrollado por google. (https://github.com/tesseract-ocr/)
- **Python:** lenguaje de programación orientado a objetos utilizado en una amplia gama de aplicaciones por su facilidad y optimización de código. (Comunidad Python: https://www.python.org/)
- **ImageMagick:** conjunto de utilidades de código abierto para mostrar, manipular y convertir imágenes, capaz de reconocer más de 200 formatos. (http://www.imagemagick.org/script/index.php**)**

- **Scikit-learn:** librería de software libre de máquinas de aprendizaje desarrolladas para Python, implementa varios algoritmos de clasificación, regresión y clustering incluyendo: support vector machines, random forests, gradient boosting, k-means, DBSCAN. (Comunidad SCIKIT LEARN: http://scikit-learn.org/)
- **QT Creator:** IDE creado por Trolltech para el desarrollo de aplicaciones con las bibliotecas Qt. (https://www.qt.io/ide/).
- **TOP:** comando Linux que permite monitorear en tiempo real el consumo de memoria y CPU de cada uno los procesos que se encuentran ejecutando, la figura 1-3 muestra el resultado del comando.

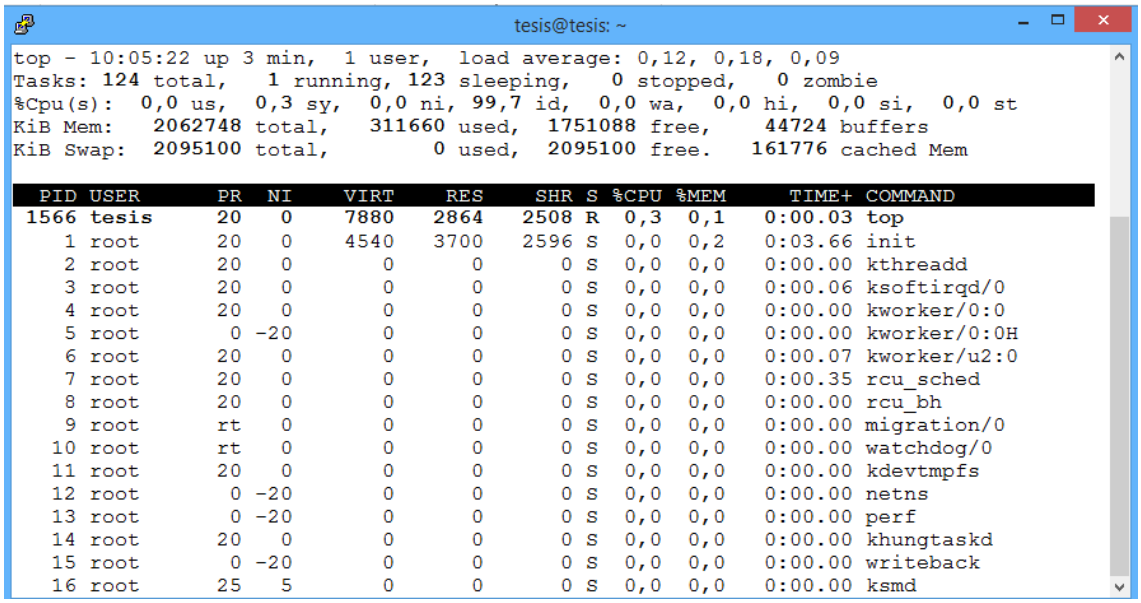


Figura 1-3 Monitoreo de Recursos con TOP
Realizado por: Arias Janeth, 2016

3.5 Población y Muestra

Para la verificación del adecuado nivel de precisión en el reconocimiento de dígitos en imágenes se seleccionó como población y muestra una serie de fotografías a medidores de energía de marca Hiking de la Empresa Eléctrica Riobamba, se realizaron 2 tomas diarias, una en la mañana y otra en la tarde durante 25 días de las cuales se seleccionaron al azar 30 muestras. La figura 2-3 muestra la imagen original del medidor:

Figura 2-3 Medidor de Energía Eléctrica Hiking
Realizado por: Arias Janeth, 2016

3.6 Planteamiento de la Hipótesis

La selección adecuada del algoritmo de reconocimiento de caracteres numéricos en imágenes digitales permitirá determinar el algoritmo con mayor grado de precisión y mínimo consumo de recursos en computadores SBC.

En las tablas 1-3 y 2-3 se detalla la operacionalización conceptual y metodológica de la hipótesis de investigación planteada

Tabla 1-3 Operacionalización Conceptual de la hipótesis de investigación

VARIABLE	TIPO	CONCEPTO
La selección adecuada del algoritmo de reconocimiento de caracteres numéricos	Simple Cualitativa Independiente	Es el conjunto de algoritmos (de aprendizaje supervisado y no supervisado) existentes para el reconocimiento de caracteres numéricos en imágenes digitales.
El grado de precisión en el reconocimiento y el consumo de recursos en computadores SBC	Compleja Cuantitativa Dependiente	Es el porcentaje de aciertos y fallos en el reconocimiento de dígitos incluidos en las imágenes digitales y el porcentaje de consumo de memoria RAM y procesamiento en computadores de tipo SBC.

Realizado por: Arias Janeth, 2016

Tabla 2-3 Operacionalización Metodológica de la hipótesis de investigación

VARIABLE	INDICADOR	TÉCNICA	INSTRUMENTO/ FUENTE
La selección adecuada del algoritmo de reconocimiento de caracteres numéricos	Tipos de algoritmos de reconocimiento de caracteres numéricos	Revisión Bibliográfica Pruebas	Algoritmos de reconocimiento de dígitos.
El grado de precisión en el reconocimiento y el consumo de recursos en computadores SBC	• % de precisión • Tiempo utilizado • Consumo de RAM • Consumo de procesamiento	Observación Analizadores de rendimiento Pruebas	Software de análisis de recursos computacionales TOP.

Realizado por: Arias Janeth, 2016

3.7 Desarrollo de la Aplicación

Una vez que las herramientas software fueron seleccionadas, se procedió a desarrollar la aplicación de visión artificial que permita verificar el adecuado funcionamiento de cada una de las técnicas de reconocimiento de dígitos seleccionadas, la figura 3-3 muestra las fases involucradas en esta aplicación.

Una vez que la fotografía del medidor de energía eléctrica es tomada, esta pasa por una serie de operaciones de tratamiento de imágenes mediante funciones disponibles en la plataforma OpenCV para su acondicionamiento, de tal forma que la fase de reconocimiento de la aplicación tenga el nivel adecuado de precisión.

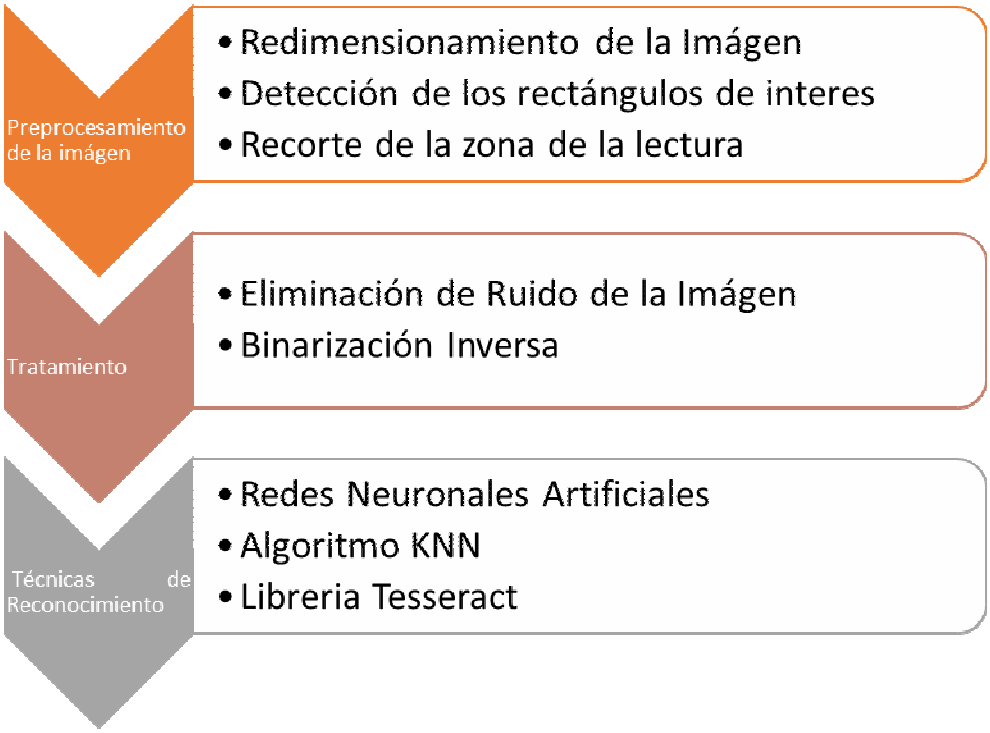


Figura 3-3 Fases para el reconocimiento de dígitos
Realizado por: Arias Janeth, 2016

A continuación se detalla cada una de las fases definidas en el pre procesamiento y tratamiento de las imágenes tomadas a los medidores de energía eléctrica.

3.7.1 Redimensionado de la Imagen

Las fotografías de las imágenes de muestra de los medidores de energía eléctrica fueron tomas en un rango de 15 cm a 30 cm con lo que el tamaño de la imagen varia notablemente, mediante varias pruebas se determinó que el tamaño adecuado para que la imagen seleccionada tenga un nivel adecuado de reconocimiento debe ser de 453 x 487 pixeles. La figura 4-3 muestra la imagen redimensionada mediante funciones de OpenCV.

Figura 4-3 Imagen redimensionada del Medidor
Realizado por: Arias Janeth, 2016

3.7.2 Detección de Zonas de Interés

Para segmentar la fotografía redimensionada del medidor de energía eléctrica, se utilizó el método de localización de objetos basado en el algoritmo para detectar blobs, los objetos dentro de la imagen se delimitaron en rectángulos que podrían contener áreas de interés como por ejemplo la zona que muestra el consumo de energía en la imagen. En esta fase del pre procesamiento se utilizan las funciones de dilatación y erosión de OpenCV. La figura 5-3 muestra el resultado de esta operación.

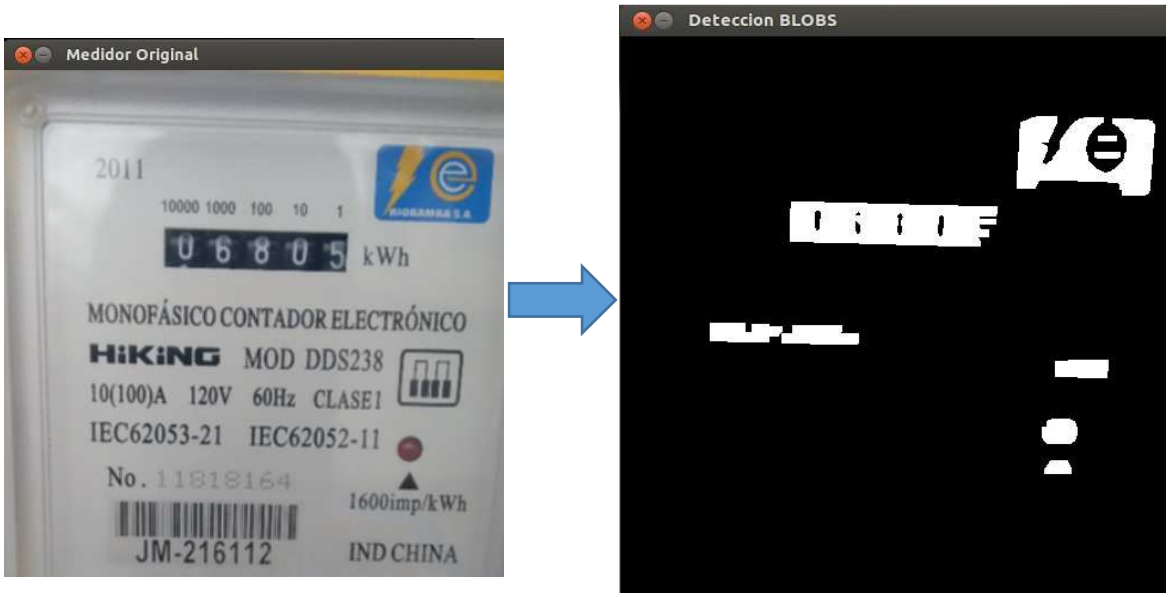


Figura 5-3 Detección de Blobs en la imagen
Realizado por: Arias Janeth, 2016

3.7.3 Recorte de la Zona de Lectura

Para finalizar esta primera etapa se recorta el blob de forma rectangular en el cual se encuentra la medición del consumo de energía, desde la posición inicial 144x170 pixeles hasta la posición final 315x201 pixeles, la figura 6-3 muestra el blob seleccionado, para esto se utiliza la función getRectSubPix.

Figura 6-3 Lectura obtenida de la imagen
Realizado por: Arias Janeth, 2016

3.7.4 Eliminación de Ruido de la imagen.

Uno de los problemas comunes que se presentan al momento de analizar imágenes obtenidas desde dispositivos de captura como las cámaras fotográficas es el ruido de compresión, cuantización y de sensibilidad del sensor de la cámara. Para resolver este problema existen varias técnicas, una de ellas es el suavizado de la imagen mediante la aplicación de filtros, para completar esta fase se utilizó el filtro gaussiano. La figura 7-3 muestra el resultado de esta operación:

Figura 7-3 Filtro Gaussiano aplicado a la imagen.
Realizado por: Arias Janeth, 2016

3.7.5 Binarización Inversa.

La binarización inversa es un proceso previo a la utilización de las funciones para el reconocimiento de patrones, que se basan en la detección de puntos blancos sobre fondos negros. Debido a que los caracteres de la imagen de la figura 7-3 son blancos en fondo negro, es necesario binarizar la imagen y posteriormente hallar su complemento.

La función de OpenCV que nos permite binarizar la imagen es threshold(), con los argumentos CV_THRESH_OTSU+CV_THRESH_BINARY_INV se aplica el método de Otsu y la binarización inversa. La figura 8-3 muestra este resultado.

Figura 8-3 Binarización Inversa de la imagen
Realizado por: Arias Janeth, 2016.

En el anexo # 1, se detalla el código completo en C++ y Opencv desarrollados para realizar las tareas del pre procesamiento de las fotografías tomadas a los medidores de energía eléctrica que anteriormente se indicaron.

3.7.6 Reconocimiento mediante Redes Neuronales Artificiales.

Como se mencionó en el apartado 2.3.3 las Redes Neuronales Artificiales (RNA) tienen un campo de aplicación muy extenso en el desarrollo de aplicaciones de inteligencia y visión artificial.

Dada la naturaleza del presente trabajo de investigación, y considerando las características de la tabla 3-3, se decidió utilizar una red neuronal de retropropagación. Este tipo de redes presenta características favorables como la gran cantidad de datos con los que pueden trabajar, el número variable de capas, una amplia variedad de funciones de activación y la capacidad de clasificar patrones no lineales.

Tabla 3-3 Características de las Redes Neuronales

Tipo de RNA	Aplicaciones	# Capas	Funciones de Activación
Perceptrón	Clasificación de Patrones Sencillos, Predicción	2	Tangente hiperbólica, Sigmoidea. umbralización
Adeline	Filtros de Ruido, Filtros Adaptativos	2	Umbralización
Madaline	Filtros de Ruido, Filtros Adaptativos	2	Umbralización
Retropropagación	Clasificación patrones complejos, minería de datos	N	Tangente hiperbólica, Sigmoidea. umbralización

Realizado por: Arias Janeth, 2016
Fuente: (Medina Sanchez, 2015)

Un factor muy importante para el éxito o fracaso de la red neuronal en el reconocimiento de caracteres es la correcta selección de los datos para su entrenamiento. El objetivo de la red neuronal en el presente trabajo de investigación, es identificar los caracteres numéricos del consumo que están contenidos en las imágenes de los medidores de energía eléctrica.

Para el entrenamiento de la red neuronal diseñada, se utilizó la base de datos de imágenes de caracteres alfanuméricos desarrollada por la Universidad de Surrey del Reino Unido, que está disponible de manera gratuita en: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/ ; y que contiene 74000 caracteres, entre los que se encuentran:

- 64 clases de caracteres (0-9, A-Z, a-z)
- 7705 caracteres obtenidos a partir de imágenes naturales
- 3410 caracteres dibujados a mano utilizando una TabletPC
- 62992 Caracteres sintetizados de fuentes de computadora

El vector de entrada de la red neuronal está formado a partir de la imagen redimensionada y binarizada. El mismo está compuesto de 626 posiciones de la matriz de 25x25 más la entrada igual a 1 del bias (neurona siempre activa para el aprendizaje). La figura 9-3 muestra uno de los dígitos de entrada para su reconocimiento en la red neuronal diseñada.

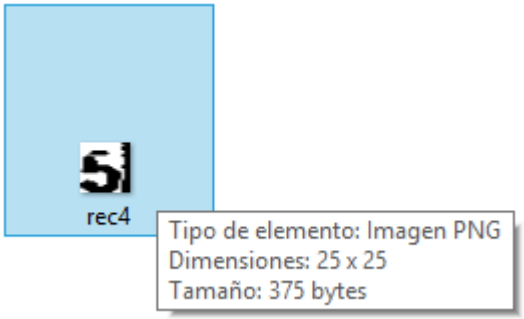


Figura 9-3 Dígito de entrada a la RNA
Realizado por: Arias Janeth, 2016

Luego de realizar varias pruebas con el número de capas de la red neuronal así como con el número de neuronas de cada una de estas, se determinó que una RNA de 3 capas es la que mejor tiempo de convergencia presentaba. La tabla 4-3 muestra los componentes de esta red neuronal.

Tabla 4-3 Componentes RNA Diseñada

Tipo de Capa	No. Capas	No. De Neuronas
Entrada	1	100
Oculta	1	250
Salida	1	37

Realizado por: Arias Janeth, 2016

El anexo # 2 muestra en detalle el programa implementado para el reconocimiento de dígitos mediante RNA.

3.7.7 Reconocimiento mediante KNN.

KNN es un algoritmo de aprendizaje supervisado, que necesita ser entrenado con una muestra de los dígitos que se desean reconocer, para cumplir con esta tarea el algoritmo encuentra el valor “K” más cercano de los datos de entrenamiento con respecto al valor que está intentando reconocer utilizando la distancia euclidiana entre dos valores.

Para el reconocimiento de la lectura del consumo del medidor de energía eléctrica, los datos de entrenamiento del algoritmo KNN estuvieron compuesto por 120 imágenes de 25x30 pixeles.

La Figura 10-3 muestra la imagen de los datos de entrenamiento.

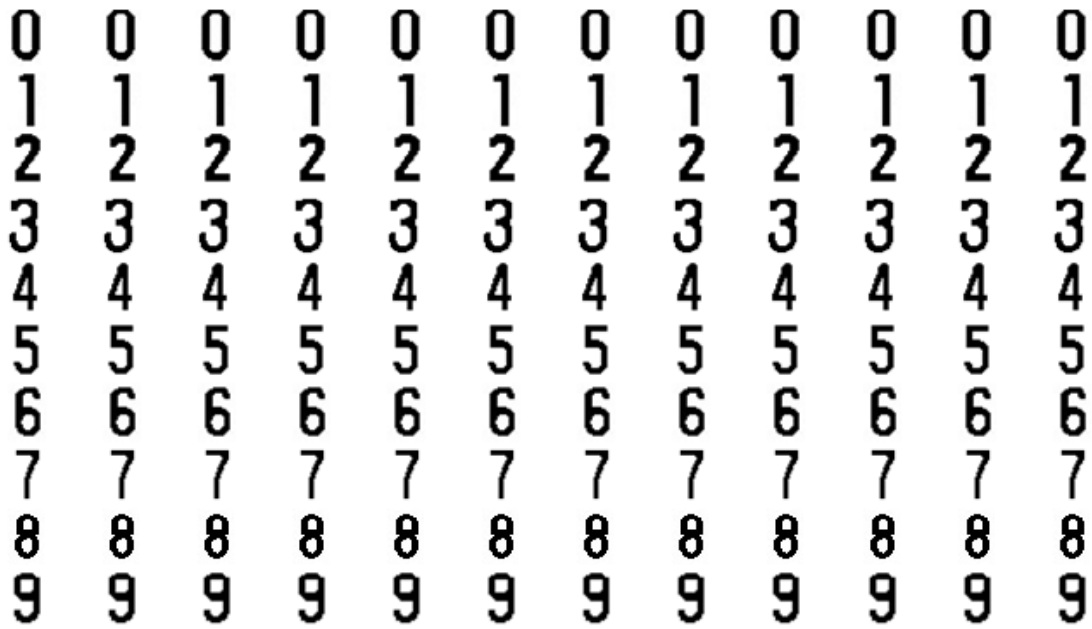


Figura 10-3 Datos de entrenamiento KNN
Realizado por: Arias Janeth, 2016

Para la generación de los datos de entrenamiento se implementó un programa en Python (Anexo #3) que permite que mediante la entrada por el teclado se identifique cada uno de los números de la figura anterior. La figura 11-3 muestra el procedimiento que se siguió.

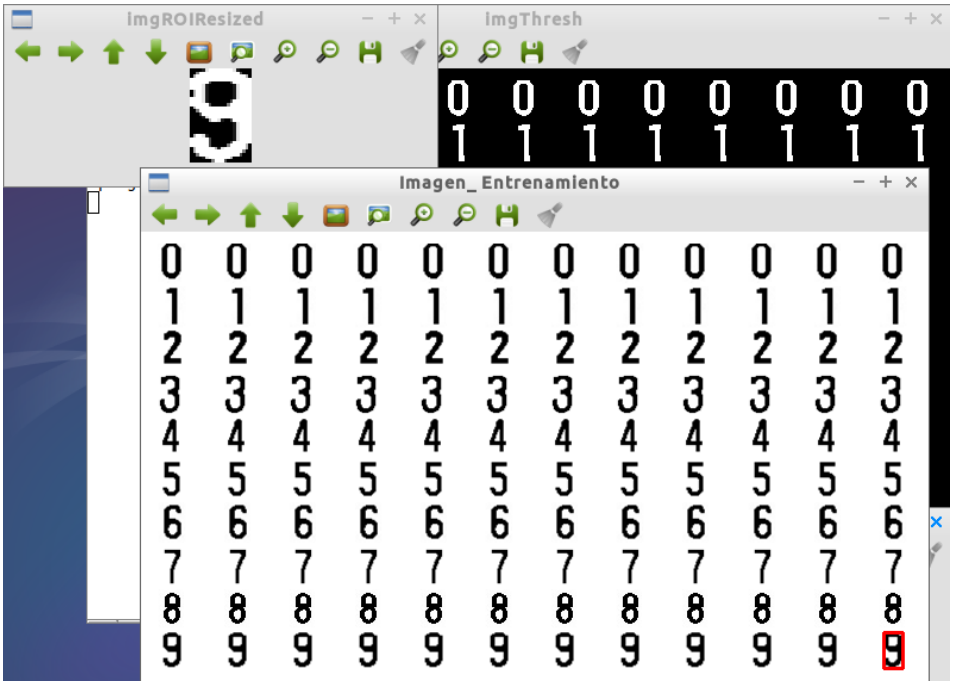


Figura 11-3 Secuencia de Entrenamiento KNN
Realizado por: Arias Janeth, 2016

Cada uno de los dígitos se redimensiona a un tamaño de 20x30 pixeles, es decir cada número se almacena en una matriz de 600 puntos. El resultado de este entrenamiento se almacena en los dos archivos que se muestran en la figura 12-3 y que contienen la secuencia 72000 bits de la imagen con los dígitos entrenados.

```
tesis@tesis: ~/knntesis
tesis@tesis:~/knntesis$ ls -lh *.txt
-rw-rw-r-- 1 tesis tesis 3,0K nov  5 19:03 clasificaciones.txt
-rw-rw-r-- 1 tesis tesis 1,8M nov  5 19:03 imagenes_planas.txt
tesis@tesis:~/knntesis$
```

Figura 12-3 Archivos Resultado del entrenamiento KNN

Realizado por: Arias Janeth, 2016

Para verificar el reconocimiento mediante el algoritmo KNN, se implementó otro programa en Python (Anexo # 4) que utiliza la función Knearest() de OpenCV, y que determina los valores más cercanos de los datos contenidos en el archivo imagenes_planas.txt y el recorte del consumo del medidor, que se obtuvo en la fase anterior. La figura 13-3 muestra el resultado de esta operación.

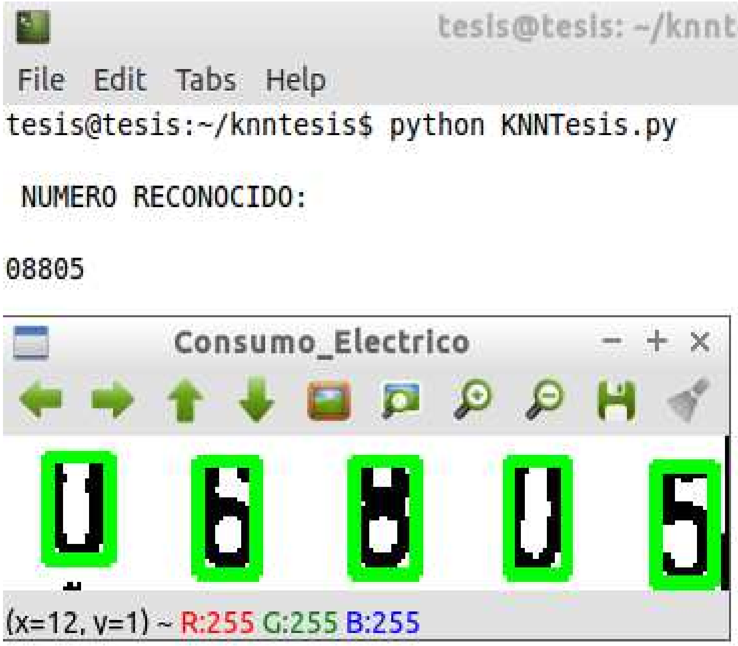


Figura 13-3 Resultado del Reconocimiento con KNN

Realizado por: Arias Janeth, 2016

3.7.8 Reconocimiento mediante Tesseract.

De manera similar a KNN, en Tesseract es muy importante realizar la generación de los datos de entrenamiento, para esto con la imagen definida de los dígitos se genera el archivo de extensión box, la figura 14-3 muestra el procedimiento realizado.

```
tesis@ubuntu: ~/traintess1
tesis@ubuntu:~/traintess1$
tesis@ubuntu:~/traintess1$
tesis@ubuntu:~/traintess1$ tesseract digitosmedidor.tif digitos.medidor.tesis.exp0 -l eng batch.nochop makebox
Tesseract Open Source OCR Engine v3.02.02 with Leptonica
TIFFReadDirectory: Warning, Unknown field with tag 769 (0x301) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 771 (0x303) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20752 (0x5110) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20753 (0x5111) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20754 (0x5112) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 769 (0x301) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 771 (0x303) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20752 (0x5110) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20753 (0x5111) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20754 (0x5112) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 769 (0x301) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 771 (0x303) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20752 (0x5110) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20753 (0x5111) encountered.
TIFFReadDirectory: Warning, Unknown field with tag 20754 (0x5112) encountered.
tesis@ubuntu:~/traintess1$
```

Figura 14-3 Generación del archivo BOX de Tesseract
Realizado por: Arias Janeth, 2016

Con los archivos generados en el paso anterior se construye los datos en entrenamiento, para esto usamos el editor jTessBoxEditor, como se muestra en la figura 15-3.

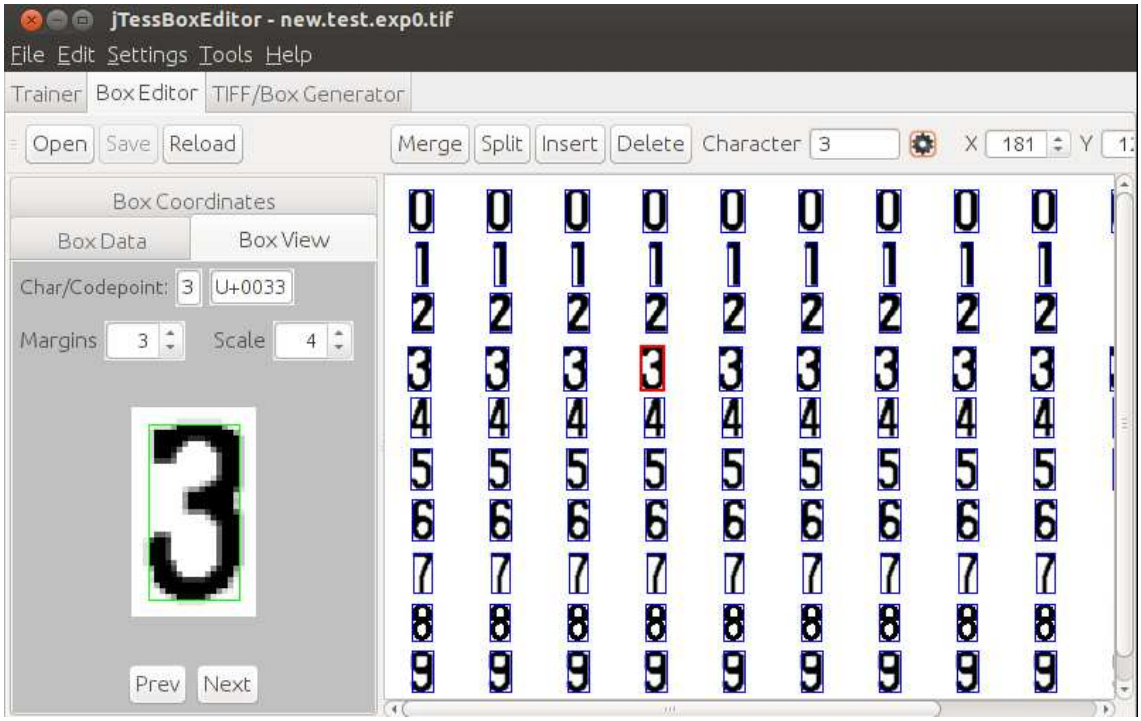


Figura 15-3 Generación de los datos de entrenamiento para Tesseract
Realizado por: Arias Janeth, 2016

Para verificar el reconocimiento de la lectura del medidor mediante el OCR Tesseract, se desarrolló un programa en C++ (Anexo # 5), que utiliza la mencionada librería y en la que se configura que los datos a utilizar para el reconocimiento sean los generados con el procedimiento anteriormente descrito, la figura 16-3 muestra el resultado de esta prueba:

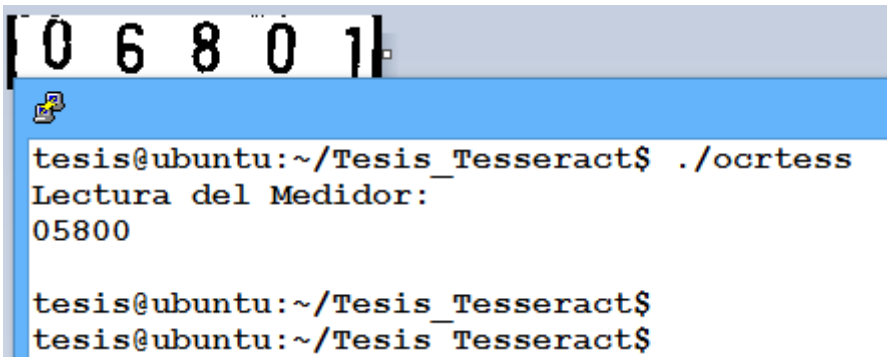


Figura 16-3 Reconocimiento del Consumo Eléctrico mediante Tesseract
Realizado por: Arias Janeth, 2016

CAPITULO IV

RESULTADOS

El presente capitulo muestra los resultados obtenidos en cada una de las pruebas realizadas con el objetivo de determinar la técnica de reconocimiento de dígitos en imágenes más adecuada que maximice la precisión de reconocimiento y minimice el consumo de recursos utilizados en esta tarea.

4.1 Resultados y discusión

Para determinar la técnica más adecuada en el reconocimiento de dígitos en imágenes se establecieron tres escenarios de pruebas con 30 muestras ponderadas de imágenes tomadas a medidores de energía eléctrica, los escenarios planteados se detallan a continuación:

- Escenario 1: Reconocimiento mediante KNN
- Escenario 2: Reconocimiento mediante Tesseract
- Escenario 3: Reconocimiento mediante Redes Neuronales Artificiales

En los escenarios planteados, se mide los efectos que la variable independiente produce en la variable dependiente, para esto los parámetros que se establecen como indicadores son los siguientes:

- **I1. Precisión**: Relación entre el número de dígitos reconocidos sobre el número total de dígitos en la imagen original.
- **I2. Tiempo Procesamiento**: Tiempo en segundos que se tarda cada una de las técnicas seleccionadas en realizar el proceso de reconocimiento.
- **I3. Consumo RAM**: Cantidad de memoria RAM del computador SBC utilizada para el proceso de reconocimiento de cada una de las técnicas.
- **I4. Consumo CPU**: Porcentaje de consumo de CPU del computador SBC utilizado para
- el proceso de reconocimiento de cada una de las técnicas.

Los datos que se obtienen de estas pruebas son analizados y comparados entre sí para posteriormente comprobar la hipótesis planteada.

A continuación se realiza la tabulación de los datos de cada uno de los indicadores que se obtuvieron en las pruebas realizadas con cada uno de los escenarios planteados.

4.2 Precisión de Reconocimiento de las Técnicas

Para el análisis de la precisión del reconocimiento de los dígitos en las imágenes tomadas a los medidores de energía eléctrica, se utilizó la fórmula que a continuación se detalla:

$$\%Precisión = \frac{\#\ Dígitos\ Reconocidos}{\#\ Dígitos\ Totales\ en\ la\ Imágen} * 100$$

En el caso de los medidores seleccionados el número total de dígitos es 3.2, como se puede observar en la figura 2-3.

Los resultados obtenidos de este indicador se muestran a continuación (Tabla 1-4).

Tabla 1-4 Porcentaje de Precisión de cada técnica

INDICADOR: PORCENTAJE DE PRECISIÓN						
	KNN		TESSERACT		RNA	
No. Muestra	Dígitos Reconocidos	% Precisión	Dígitos Reconocidos	% Precisión	Dígitos Reconocidos	% Precisión
1	4	80	3	60	3	60
2	1	20	2	40	1	20
3	0	0	0	0	1	20
4	2	40	5	100	0	0
5	4	80	4	80	1	20
6	4	80	0	0	1	20
7	2	40	3	60	2	40
8	3	60	0	0	2	40
9	0	0	0	0	1	20
10	2	40	4	80	1	20
11	1	20	0	0	0	0
12	2	40	4	80	0	0
13	1	20	1	20	1	20
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	4	80	0	0	2	40
17	2	40	0	0	1	20
18	3	60	2	40	1	20
19	2	40	0	0	1	20
20	1	20	0	0	2	40
21	1	20	0	0	2	40
22	3	60	0	0	2	40
23	1	20	0	0	0	0

24	5	100	3	60	1	20
25	5	100	1	20	2	40
26	3	60	2	40	1	20
27	4	80	4	80	0	0
28	5	100	0	0	1	20
29	4	80	1	20	2	40
30	5	100	5	100	1	20
Promedio	**2,47**	**49,33**	**1,47**	**29,33**	**1,10**	**22,00**
Valor Mínimo	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
Valor Máximo	**5,00**	**100,00**	**5,00**	**100,00**	**2,00**	**40,00**

Realizado por: Arias Janeth, 2016

Como se puede observar en la tabla 1-4, la técnica KNN es la que mayor porcentaje de precisión tiene en el reconocimiento de los dígitos alcanzando un 49,33% frente al 29,33% y 22% de las técnicas TESSERACT y RNA, respectivamente.

La gráfica 1-4, muestra claramente que la técnica KNN tiene un mayor número de aciertos del 100% del reconocimiento de los dígitos de la imagen procesada.

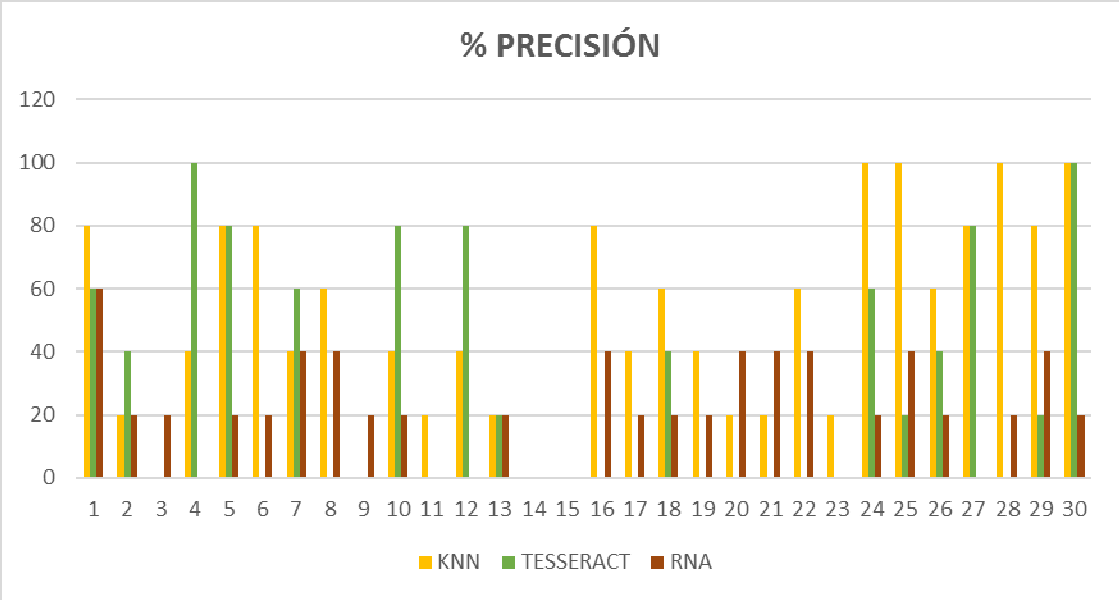


Gráfico 1-4 Porcentaje de Precisión de las Técnicas Seleccionadas
Realizado por: Janeth Arias G. 2016

4.3 Tiempo de procesamiento de cada Algoritmo

Para el análisis del tiempo empleado por cada una de las técnicas en el reconocimiento, se utilizó el comando de Linux: time, que muestra el tiempo en segundos utilizado por un proceso específico, en nuestro caso el tiempo utilizado por los programas en Python y C++ implementados.

La tabla 2-4 muestra los resultados obtenidos para este indicador

Tabla 2-4 Tiempo de procesamiento utilizado por cada técnica

INDICADOR: TIEMPO DE PROCESAMIENTO			
No. Muestra	**KNN**	**TESSERACT**	**RNA**
1	1,276	0,072	0,32
2	1,688	0,04	0,34
3	1,556	0,064	0,305
4	0,736	0,08	0,292
5	0,58	0,056	0,334
6	1,196	0,04	0,35
7	0,788	0,048	0,281
8	1,584	0,04	0,333
9	1,188	0,056	0,312
10	1,808	0,068	0,304
11	0,956	0,04	0,255
12	0,988	0,052	0,316
13	2,292	0,076	0,316
14	0,864	0,044	0,246
15	0,776	0,056	0,217
16	1,086	0,036	0,291
17	1,656	0,068	0,294
18	1,044	0,048	0,283
19	1,972	0,052	0,271
20	1,336	0,072	0,284
21	2,072	0,08	0,291
22	1,82	0,044	0,288
23	0,708	0.064	0,249
24	0,7	0.060	0,301
25	0,964	0,056	0,307
26	0,8	0,06	0,312
27	1,772	0,052	0,299
28	0,84	0,068	0,3
29	0,936	0,156	0,287
30	0,48	0,072	0,297
Valor Mínimo	**0,48**	**0,04**	**0,22**
Valor Máximo	**2,29**	**0,16**	**0,35**

Realizado por: Arias Janeth, 2016

Como se puede observar en la tabla 2-4, la técnica TESSERACT es la que menor tiempo de procesamiento utilizas en el reconocimiento de los dígitos con 0,04 segundos frente a los 0,322

y 0,48 segundos de las técnicas RNA y KNN, respectivamente. De la misma forma se evidencia que KNN utiliza el tiempo más alto para el reconocimiento alcanzando los 2,29 segundos.

La gráfica 4.2 muestra claramente que las técnicas KNN y RNA son las que mayor tiempo de procesamiento utilizan en el reconocimiento de los dígitos.

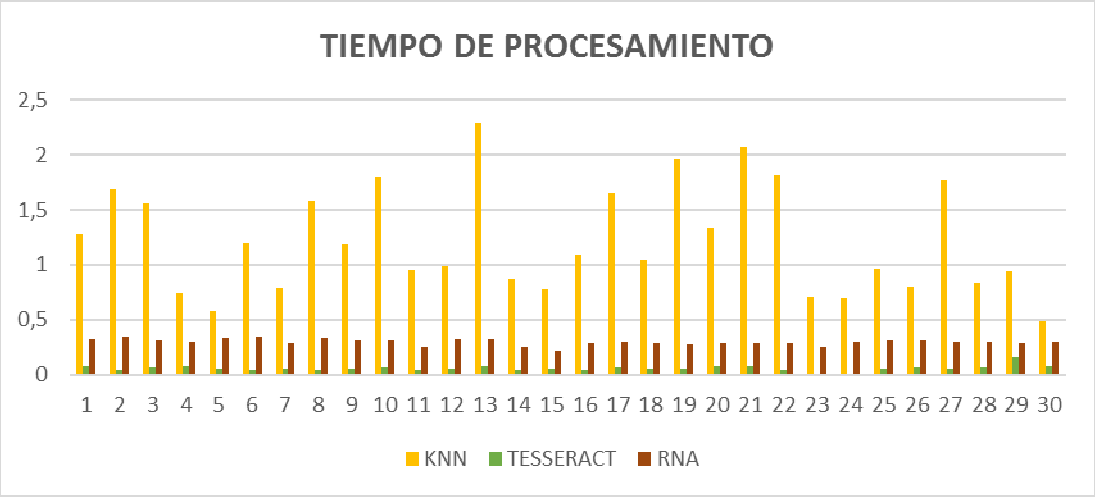


Gráfico 2-4 Tiempo de Procesamiento utilizado por las Técnicas Seleccionadas
Realizado por: Arias Janeth, 2016

4.4 Consumo de Memoria RAM de cada Algoritmo

Para determinar la cantidad de memoria RAM utilizada por cada una de las técnicas en el reconocimiento, se utilizó el comando de Linux: top, que muestra en tiempo real la cantidad en Megabytes de este recurso utilizado por los programas en Python y c++ implementados.

A continuación se muestra los resultados obtenidos para este indicador (Tabla 3-4).

Tabla 3-4 Consumo de Memoria RAM de cada técnica

INDICADOR: CONSUMO MEMORIA RAM			
No. Muestra	**KNN**	**TESSERACT**	**RNA**
1	15,7	52,8	4,56
2	15,7	52,8	4,56
3	15,7	52,8	4,56
4	15,7	52,8	4,56
5	15,7	52,8	4,56
6	15,7	52,8	4,56
7	15,7	52,8	4,56
8	15,7	52,8	4,56

9	15,7	52,8	4,56
10	15,7	52,8	4,56
11	15,7	52,8	4,56
12	15,7	52,8	4,56
13	15,7	52,8	4,56
14	15,7	52,8	4,56
15	15,7	52,8	4,56
16	15,7	52,8	4,56
17	15,7	52,8	4,56
18	15,7	52,8	4,56
19	15,7	52,8	4,56
20	15,7	52,8	4,56
21	15,7	52,8	4,56
22	15,7	52,8	4,56
23	15,7	52,8	4,56
24	15,7	52,8	4,56
25	15,7	52,8	4,56
26	15,7	52,8	4,56
27	15,7	52,8	4,56
28	15,7	52,8	4,56
29	15,7	52,8	4,56
30	15,7	52,8	4,56
Valor Mínimo	**15,70**	**52,80**	**4,56**
Valor Máximo	**15,70**	**52,80**	**4,56**

Realizado por: Arias Janeth, 2016

Como se puede observar en la tabla 3-4 el consumo de memoria RAM es constante en el procesamiento de cada una de las 30 muestras con cada una de las técnicas seleccionadas, de la misma forma se determina que el uso de Redes Neuronales Artificiales en el reconocimiento de los dígitos consume la menor cantidad de memoria RAM con 4.56 megabytes frente a los 52,8 y 15,7 megabytes consumidos por TESSERACT y KNN respectivamente.

En la gráfica 3-4 se puede observar que TESSERACT es la técnica que mayo consumo de memoria RAM realiza en el reconocimiento de los dígitos.

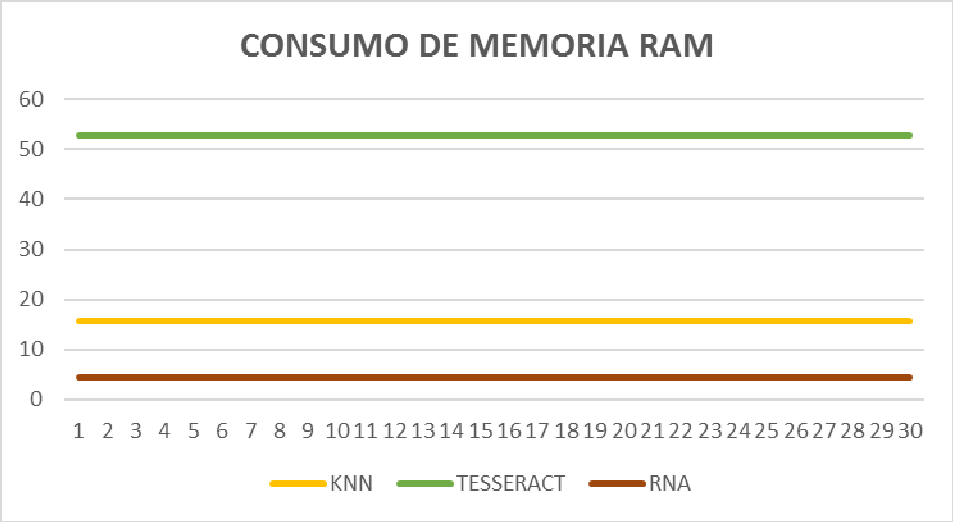


Gráfico 3-4 Tiempo de Procesamiento utilizado por las Técnicas Seleccionadas
Realizado por: Arias Janeth, 2016

4.5 Consumo de CPU de cada Algoritmo

Para determinar el porcentaje de uso de CPU utilizado por cada una de los algoritmos en el reconocimiento, se utilizó el comando de Linux top, que muestra en tiempo real el consumo de este recurso utilizado por los programas en Python y C++ implementados.

A continuación se muestra los resultados obtenidos para este indicador (Tabla 4-4)

Tabla 4-4 Consumo de Memoria RAM de cada técnica

INDICADOR: % CONSUMO DE CPU			
No. Muestra	**KNN**	**TESSERACT**	**RNA**
1	12,1	2,3	20,4
2	10,6	3,3	22,1
3	11,9	3	21,8
4	12	2,7	15,2
5	11,8	2,3	15,4
6	11,3	2,3	19,1
7	11,9	2,6	13,6
8	11,2	2,7	23
9	11,9	3,3	15,6
10	12,1	2,7	22,5
11	11,2	2,7	15,8
12	12,1	2,3	13,9

13	12	2,3	15,5
14	12,1	2,3	15,4
15	10,3	3,3	16,4
16	11,8	2,3	22,2
17	11,4	2,7	17,8
18	11,7	2,7	14,1
19	11,3	3,3	21,8
20	11,9	2,3	22,6
21	11,2	2,7	20,6
22	11,3	2,3	15,8
23	11,9	2,6	20,8
24	11,4	2,7	15,7
25	11,3	2,7	13,2
26	11,9	2,6	22
27	12,1	2,3	21,9
28	11,1	3	14,4
29	12,2	2,9	15,7
30	12,3	2,3	16,1
Valor Mínimo	**10,30**	**2,30**	**13,20**
Valor Máximo	**12,30**	**3,30**	**23,00**

Realizado por: Arias Janeth, 2016

Como se observa en la tabla 4-4 el uso de Redes Neuronales Artificiales en el reconocimiento de los dígitos consume la mayor cantidad de procesamiento con el 23,01% de CPU en contraste al 3,30% y a los 12,30% consumidos por TESSERACT y KNN respectivamente.

La gráfica 4-4 muestra que la técnica TESSERACT es la de menor consumo de CPU en el reconocimiento de los dígitos.

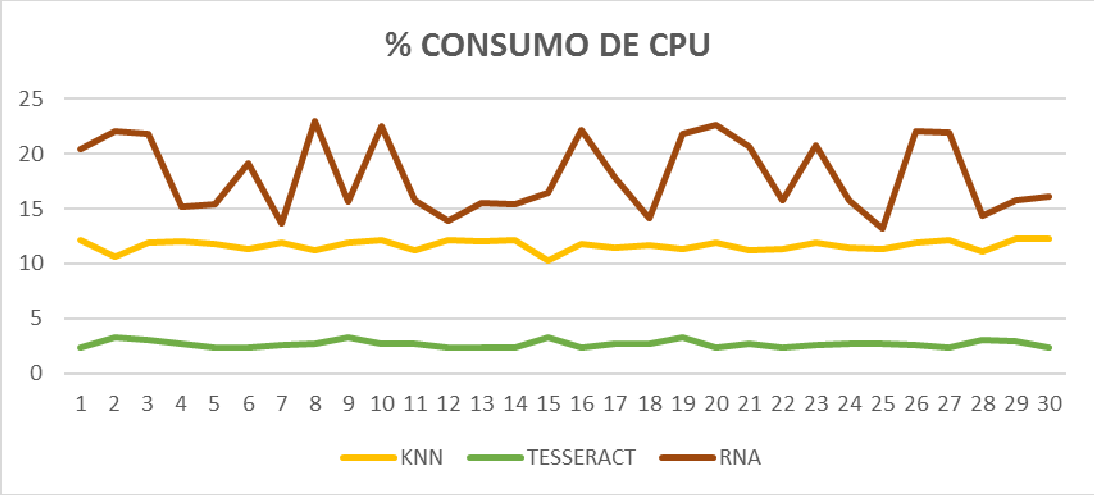


Gráfico 4-4 Porcentaje de Consumo de CPU utilizado por las Técnicas Seleccionadas
Realizado por: Arias Janeth, 2016

4.6. Resumen del análisis de los Indicadores

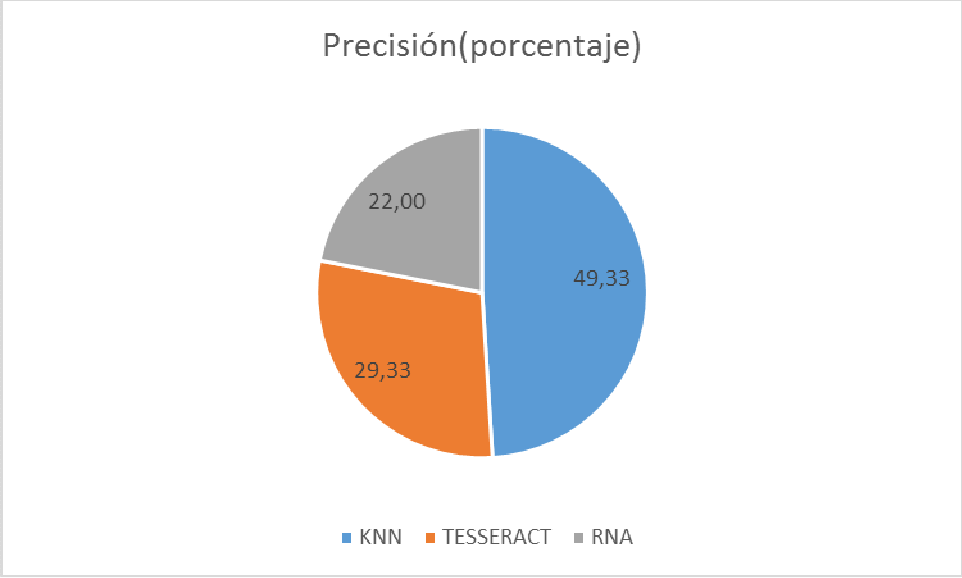
A continuación en la tabla 5-4 se muestra el promedio de cada uno de los indicadores obtenidos por cada una de las técnicas seleccionadas para el reconocimiento de dígitos las imágenes tomadas a los medidores de energía eléctrica.

Tabla 5-4 Promedio de cada indicador cada técnica

	TECNICAS DE RECONOCIMIENTO		
Indicador	**KNN**	**TESSERACT**	**RNA**
I1.Precisión(porcentaje)	49,33	29,33	22,00
I2. Tiempo Procesamiento(segundos)	1,22	0,06	0,30
I3. Consumo RAM(megabytes)	15,70	52,80	4,56
I4. Consumo CPU(porcentaje)	11,64	2,65	18,01

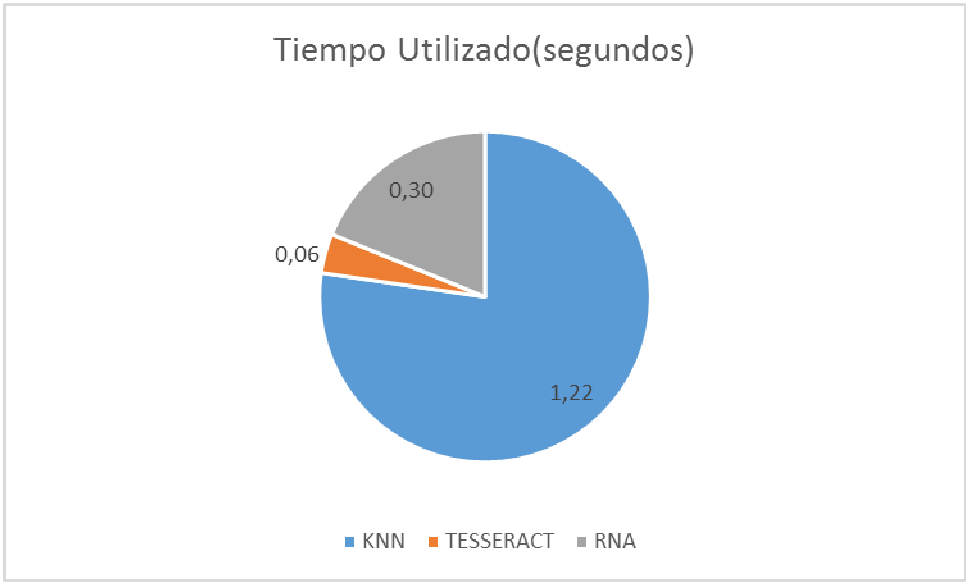
Realizado por: Arias Janeth, 2016

Del análisis del indicador **I1. Precisión** que se muestra en la gráfica 5-4, se puede concluir que la técnica que mayor precisión tiene a la hora de realizar el reconocimiento de los dígitos en las imágenes tomadas a los medidores de energía eléctrica es KNN.



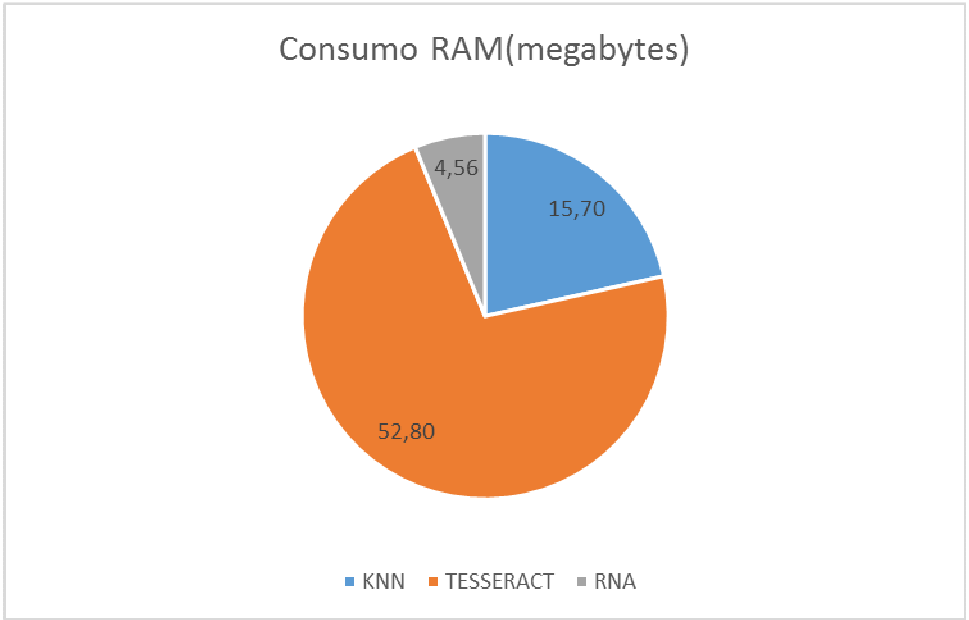
Gráfica 5-4 Porcentaje de Precisión en el Reconocimiento de Dígitos
Realizado por: Arias Janeth, 2016

De la misma forma, del análisis del indicador **I2. Tiempo Procesamiento** que se muestra en la gráfica 6-4, se puede concluir que la técnica que menor tiempo utiliza para realizar el reconocimiento de los dígitos en las imágenes tomadas a los medidores de energía eléctrica es TESSERACT.



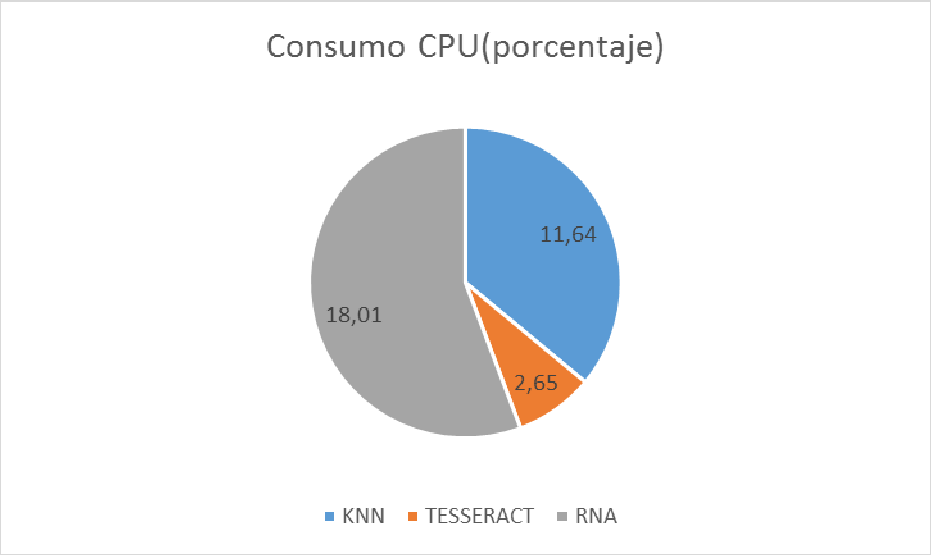
Gráfica 6-4 Tiempo de Procesamiento utilizado en el Reconocimiento de Dígitos
Realizado por: Arias Janeth, 2016

Al analizar la información del indicador **I3. Consumo RAM** que se muestra en la gráfica 7-4, se puede concluir que la técnica que menor cantidad de memoria RAM necesita para realizar el reconocimiento de los dígitos en las imágenes tomadas a los medidores de energía eléctrica es RNA.



Gráfica 7-4 Consumo de Memoria RAM utilizado en el Reconocimiento de Dígitos
Realizado por: Arias Janeth, 2016

Finalmente, en el análisis del indicador **I4. Consumo CPU** que se muestra en la gráfica 8-4, se puede concluir que la técnica que menor consumo de CPU necesita para realizar el reconocimiento de los dígitos en las imágenes tomadas a los medidores de energía eléctrica es TESSERACT.



Gráfica 8-4 Consumo de CPU utilizado en el Reconocimiento de Dígitos
Realizado por: Arias Janeth, 2016

4.7 Comprobación de la Hipótesis de investigación

Para realizar la comprobación de la hipótesis de investigación es necesario primero verificar si los datos de los indicadores de cada una de las técnicas cumple con la condición de normalidad, esta condición nos permitirá determinar si es necesario utilizar una prueba estadística paramétrica o no paramétrica.

Para realizar la prueba de normalidad se utilizó el software estadístico MINITAB versión 17, las figuras 1-4, 2-4 y 3-4 muestran el resultado de esta prueba para el indicador **I1. Precisión**, con cada una de las técnicas de reconocimiento de dígitos.

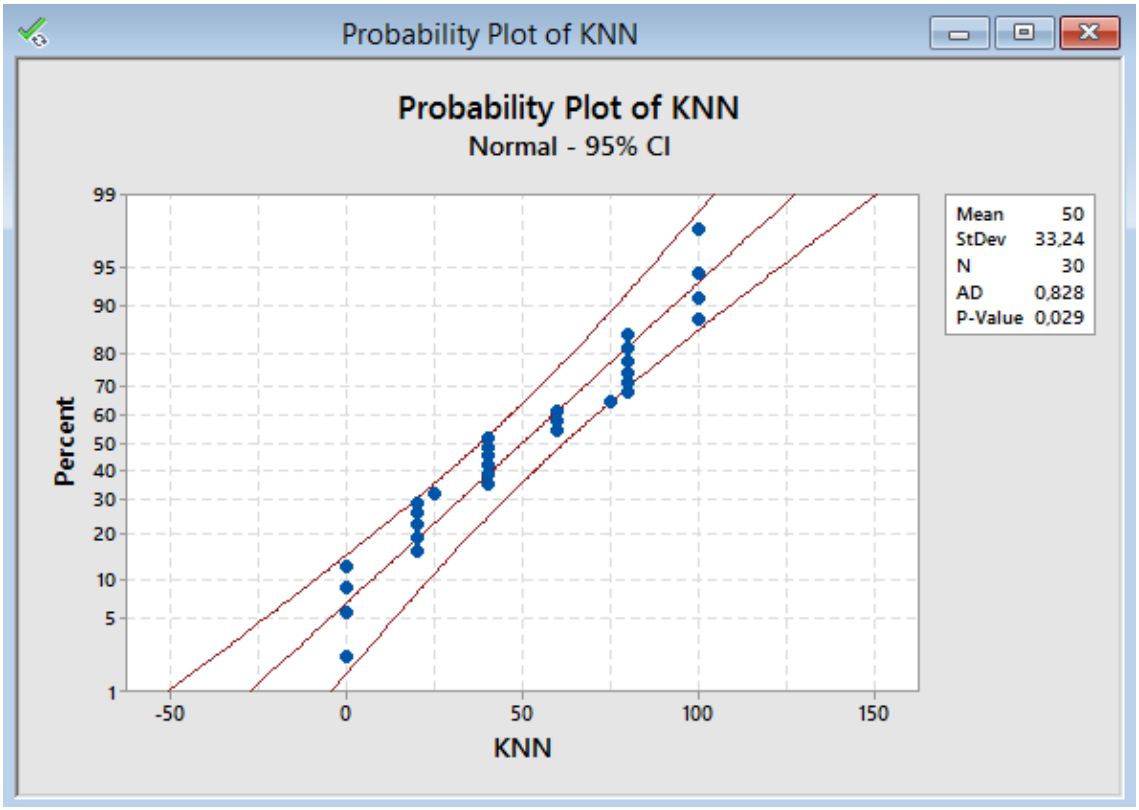
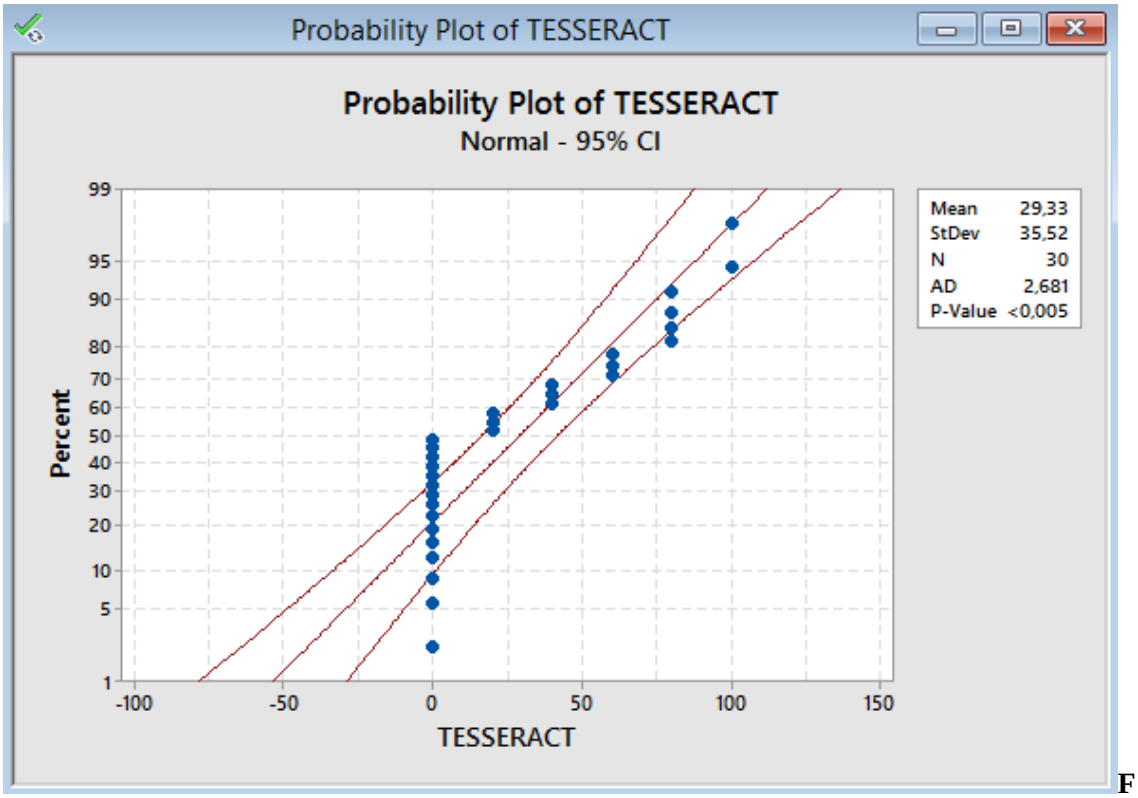


Figura 1-4 Prueba de Normalidad del indicador I1 – KNN
Realizado por: Arias Janeth, 2016



F

Figura 2-4 Prueba de Normalidad del indicador I1 – TESSERACT
Realizado por: Arias Janeth, 2016

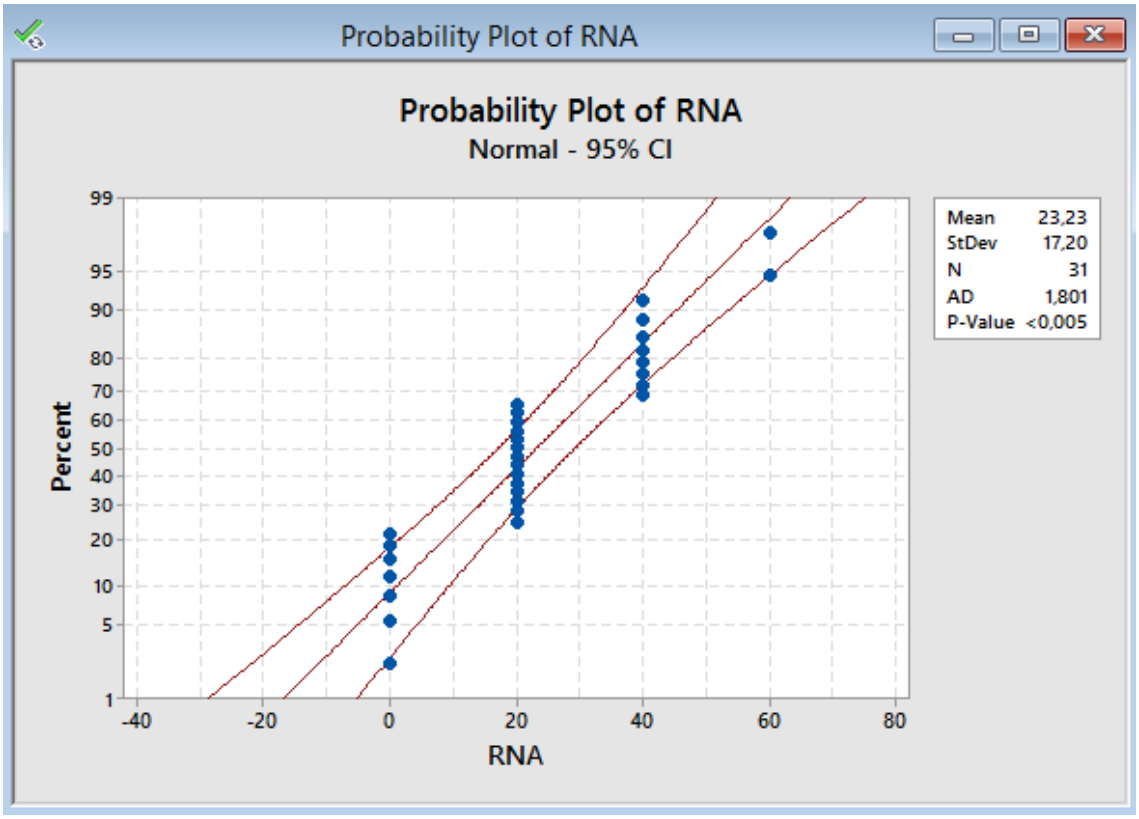


Figura 3-4 Prueba de Normalidad del indicador I1 – RNA
Realizado por: Arias Janeth, 2016

Para determinar o no la normalidad de los datos, se utilizó el estadístico de Anderson-Darling, (Walpole, 1999) que plantea las hipótesis que se muestra en la tabla 6-4:

Tabla 6-4 Hipótesis de Normalidad de Datos

Hipótesis	**Enunciado**
H0 (Nula)	Los datos del indicador I1. Precisión provienen de una distribución normal.
H1 (Alternativa)	Los datos del indicador I1. Precisión NO provienen de una distribución normal.

Realizado por: Arias Janeth, 2016

Para la comprobación de las hipótesis planteadas se debe cumplir la siguiente condición: "Si el estadístico calculado es mayor que 0.751, entonces se acepta H1 y se rechaza H0".

La tabla 7-4 muestra el estadístico de Anderson-Darling calculado para cada una de las técnicas de reconocimiento de dígitos.

Tabla 7-4 Valores de la Prueba de Normalidad

Técnica de Reconocimiento	**Estadístico Anderson Darling Calculado**
KNN	0,828
TESSERACT	2,681
RNA	1,801

Realizado por: Arias Janeth, 2016

Como se puede observar en la tabla 7-4 todos los valores calculados son mayores que 0.751, por lo que se concluye que los datos del indicador I1.Precisión no provienen de una distribución normal, por lo que, para la demostración de la hipótesis de investigación es necesario utilizar una prueba estadística NO paramétrica.

De la amplia gama de pruebas no paramétricas existentes, se seleccionó la de KRUSKAL-WALLIS porque cumple las siguientes condiciones:

- Las muestras son independientes
- Existen más de dos técnicas a comparar

La tabla 8-4 detalla las hipótesis de investigación que se plantean:

Tabla 8-4 Hipótesis de Investigación Planteadas

Hipótesis	**Enunciado**
H0 (Nula)	El nivel de precisión en el reconocimiento de los dígitos es igual en cada una de las técnicas seleccionadas
H1 (Alternativa)	El nivel de precisión en el reconocimiento de los dígitos NO es igual en cada una de las técnicas seleccionados

Realizado por: Arias Janeth, 2016

A continuación en la tabla 9-4 y 10-4, se muestran los resultados de la prueba no paramétrica obtenidos para el indicador **I1. Precisión**.

Tabla 9-4 Resultados de la Prueba de Kruskal-Wallis

Chi-Cuadrado	10,814
Grados de Libertad	2
P-Valor	0,004

Realizado por: Arias Janeth, 2016

Como se puede observar en la tabla 9-4, el P-valor es menor que 0,05 por lo que se rechaza la hipótesis nula (H0) y se acepta la hipótesis alternativa (H1) de investigación, concluyendo que: **"El nivel de precisión en el reconocimiento de los dígitos NO es igual en cada una de los técnicas seleccionadas."**

Para determinar la mejor técnica en el reconocimiento de dígitos, recurrimos a la información que se muestra en la tabla 10-4, en la que se detalla el rango promedio de cada una de las técnicas estudiadas, como se puede observar **KNN** presenta el mayor valor con 57,95 frente a 39,83 y 38,70 de TESSERACT y RNA respectivamente.

Tabla 10-4 Rangos Resultantes de la Prueba de Kruskal-Wallis

Algoritmo	# Muestras	Rango Promedio
KNN	30	57,95
TESSERACT	30	39,83
RNA	30	38,70

Realizado por: Arias Janeth, 2016

De esta manera concluimos que **KNN es la técnica de reconocimiento de dígitos con mayor grado de precisión y un nivel adecuado de consumo de recursos** como se indica en la gráfica 7-4 y 8-4.

4.8 Propuesta

Debido a que las fotografías tomadas a los medidores de energía eléctrica se realizaron a diferentes horas del día y diferentes distancias desde la cámara hasta el medidor, se debe trabajar en mejorar el sistema de adquisición de datos, para que la imágenes obtenidas tengan un mayor nivel de nitidez, ya que como se puedo evidenciar en la investigación este es un factor que afecta mucho al momento de realizar el reconocimiento de los caracteres numéricos.

Se propone también estudiar otras técnicas de reconocimiento de caracteres como las redes neuronales convolucionales y la técnica de aprendizaje profundo en aplicaciones de visión artificial.

CONCLUSIONES

- Se realizó un estudio detallado de los algoritmos de reconocimiento de caracteres KNN, Tesseract y Redes Neuronales Artificiales y sus principales aplicaciones en el campo del reconocimiento de patrones.

- Se desarrolló una plataforma opensource mediante el sistema operativo GNU/LINUX, la librería de visión artificial OpenCV y los lenguajes de programación Python y C++, para la evaluación del nivel de precisión en el reconocimiento de la lectura del consumo de las imágenes tomadas a medidores de energía eléctrica.

- En las pruebas realizadas para determinar la técnica de reconocimiento de caracteres numéricos con mayor grado de precisión se utilizó una muestra ponderada de treinta fotografías tomadas a medidores de energía eléctrica en las cuales el consumo se indica en un número de cinco dígitos; obteniéndose los siguientes resultados: 49,33% de precisión con el algoritmo KNN, 29.33% con TESSERACT y 22% con el uso de Redes Neuronales Artificiales, utilizando un tiempo promedio de 1.22 , 0.06 y 0.3 segundos respectivamente en cada una de los algoritmos indicados.

- En lo relacionado con el consumo de recursos utilizados para el reconocimiento de los dígitos los resultados fueron los siguientes: KNN consumió en promedio 15.7 megabytes de memoria RAM y 11.64% de consumo de CPU; TESSERACT consumió 52.8 megabytes de memoria RAM y 2.65% de CPU; finalmente las Redes Neuronales Artificiales consumieron 4.56 megabytes de memoria RAM y 18.01% de CPU.

- Para la comprobación de la hipótesis de investigación planteada: “La selección adecuada del algoritmo de reconocimiento de caracteres numéricos en imágenes digitales permitirá determinar el algoritmo con mayor grado de precisión y mínimo consumo de recursos en

computadores SBC", se utilizó la prueba estadística no paramétrica de Kruskall-Wallis debido a que los datos de los indicadores % de Precisión, Tiempo de Procesamiento, Cantidad Consumida de memoria RAM y % de uso del CPU, no cumplen con la condición de normalidad que se demostró utilizando el test estadístico de Anderson-Darling.

- De los resultados obtenidos de la prueba no paramétrica se pudo determinar que la técnica que mejor rango promedio presenta es KNN, esto se corroboró con él % de precisión que se obtuvo al momento de realizar el reconocimiento de los dígitos por cada una de los algoritmos seleccionados.

RECOMENDACIONES

- Es recomendable realizar la captura de la imagen a ser procesada durante horas del día en las que las condiciones climatológicas presentes no afectan al posterior reconocimiento de la zona de interés de la misma.

- Para mejorar la precisión en el reconocimiento de los dígitos se recomienda incrementar el tamaño de la muestra de dígitos de entrenamiento de KNN con una variedad más amplia de fuentes tipográficas.

- Como trabajos futuros se recomienda profundizar en el estudio de la técnica KNN enfocadas a diferentes aplicaciones de visión artificial en la industria, así como evaluar nuevas técnicas de reconocimiento de patrones como SVM, Deep Learning y Aprendizaje no Supervisado, que se han convertido en campos de investigación activos en muchas universidades del mundo.

- Es recomendable realizar una prueba de normalidad con los datos de los indicadores del trabajo de investigación para confirmar el uso de pruebas estadísticas paramétricas o no paramétricas en la demostración de la hipótesis.

GLOSARIO

ANDERSON-DARLING: Prueba estadística que permite verificar la normalidad o no de un conjunto de datos.

BINARIZACIÓN: Proceso que consiste en la reducción de información de una imagen digital de color a valor binarios de 1 y 0 que corresponden a los colores Blanco y Negro.

BLOB: Tamaño binario del objeto a reconocer, se refiere a un grupo de pixeles conectados dentro de la imagen binaria que se desea reconocer.

BLUR: Tipo de filtro gaussiano que elimina el ruido presente en una imagen producto de su procesamiento previo.

CPU: Unidad Central de Procesamiento del computador.

KNN: K vecinos más cercanos, técnica de clasificación de patrones supervisada desarrollada por Fix y Hodges en 1951.

KRUSKALL-WALLIS: Prueba estadística no paramétrica que compara los rangos promedios de N muestras independientes.

OPENCV: Conjunto de librerías multiplataforma de código libre orientado a aplicaciones de visión artificial desarrollado originalmente por Intel en 1999.

PNG: Formato gráfico de imágenes digitales basado en un algoritmo de compresión sin pérdida de información del mapa de bits. Este formato fue desarrollado en buena parte para solventar las deficiencias del formato GIF y permite almacenar imágenes con una mayor profundidad de contraste y otros importantes datos.

PYTHON: Lenguaje de programación interpretado con soporte de módulos para el manejo de OpenCV.

RAM: Memoria de lectura y escritura presente en el computador y que es utilizado para el funciones relacionadas con el procesamiento de imágenes.

RNA: Redes Neuronales Artificiales, una técnica de inteligencia artificial para el desarrollo de aplicaciones de visión por computador.

RASPBIAN: Distribución del sistema operativo GNU/Linux basado en Debian Wheezy (Debian 7.0) para la placa computadora (SBC) Raspberry Pi.

SBC: Computador de placa reducida para el desarrollo de sistemas electrónicos embebidos

TESSERACT: Motor de reconocimiento óptico de caracteres de código libre desarrollado inicialmente por Hewlett Packard y la Universidad de Nevada, actualmente google continua con su desarrollo.

TOP: Programa de GNU/Linux que desarrollado para monitorear en tiempo real el consumo de recursos de un computador como la memoria RAM y el CPU utilizado por un proceso específico.

BIBLIOGRAFIA

Aplicaciones de la Visión Artificial. (1993). Artificial, 1–27.

Arévalo, V. M., González, J., & Ambrosio, G. (2002). La librería de visión artificial opencv aplicación a la docencia e investigación 1, 1–6.

Bertona, L. (2005). Entrenamiento de Redes Neuronales Basado en Algoritmos Evolutivos. Buenos Aires.

Dedgaonkar, S., Chandavale, A., & Sapkal, A. (2012). Survey of Methods for Character Recognition. *Ijeit.Com*, *1*(5), 180–189. Retrieved from http://ijeit.com/vol 1/Issue 5/IJEIT1412201205_36.pdf

Dueñas, C. P. (2014). Apuntes del Curso Robótica y Visión Artificial. Retrieved from www.elai.upm.es/webantigua/spain/Asignaturas/Roboticas/InfoRobotica.htm

Flores López, R., & Fernández Fernández , J. M. (2008). *Las Redes Neuronales Artificiales. Fundamentos teóricos y aplicaciones prácticas.* La Coruña: Netbiblo, S.L.

Henno, J. (2015). *Empirical Evaluation of Approaches for Digit Recognition*. Linnaeus University. Suecia.

Hernández, S. (2010). *Metodología de la Investigación*. (M.-H. Interamericana, Ed.).

Ingelmo, A. M. (2009a). *Estudio de técnicas de caracterización de la figura humana, para su posible aplicación a problemas de reconocimiento de género*. Universidad de Jaume.

Medina Sanchez, M. D. (2015). *Desarrollo de un Sistema de Acceso para Parqueaderos y Cobro Tarifario utilizando reconocimiento de Patrones*. Escuela Superior Politécnica de Chimbrazo.

Montoya Holguín, C., Cortés Osorio, J. A., & Chaves Osorio, J. A. (2014). Sistema automático de reconocimiento de frutas basado en visión por computador. *Ingeniare: Revista Chilena de Ingeniería*, *22*(4), 504–516. http://doi.org/10.4067/S0718-33052014000400006

Pajankar, A. (2015). *Raspberry Pi Computer Vision Programming*. (P. Publishing, Ed.) (first).

Pajares, M. (2008). *Visión por computador: Imágenes Digitales y Aplicaciones. Pajares Martinsanz. ISBN: 978-970-15-1356-9. Mexico. 2008. Visión por computador: Imágenes Digitales y Aplicaciones.* (Ra-ma, Ed.).

Robert Laganiere. (2014). *OpenCV Computer Vision Application Programming Cookbook.* (Packt Publishing, Ed.).

Sánchez Fernández, C. J., & Sadonís Consuegra, V. (2009). Reconocimiento Óptico de Caracteres (OCR), 7.

Sanz, J. (2008). *Reconocimiento de objetos por descriptores de forma*. Universidad de Barcelona.

Sarthak Jain, Anant Vaibhav, L. G. . (2014). Raspberry Pi based Interactive Home Automation System through E-mail. *International Conference on Reliability, Optimization and Information Technology (ICROIT).*

Serpa Andrade, L. J. (2014). *Propuesta de un método basado en visión por computador como herramienta de apoyo en el diagnostico endoscópico*. Universidad Politécnica Salesiana.

Sobrado, A. (2003). *Sistema de Visión Artificial para el reconocimiento y manipulación de objetos utilizando un brazo Robot*. Pontificia Universidad Católica del Perú.

Walpole, R. (1999). *Probabilidad y Estadística para Ingenieros.* (P. Education, Ed.).

Warren Gay. (2014). *Mastering Raspberry Pi.* (Apress, Ed.).

Yu, Q., Cheng, H. H., Cheng, W. W., & Zhou, X. (2004). Ch OpenCV for interactive open architecture computer vision, *35*, 527–536.

ANEXOS

ANEXO # A

```
// Programa que identifica y recorta el valor numérico de la imagen del medidor
// Autor: Janeth Arias G. 2016


#include <iomanip>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <math.h>
#include <fstream>
#include <tesseract/baseapi.h>
#include <leptonica/allheaders.h>
#include <tesseract/strngs.h>


using namespace std;

using namespace cv;

bool verif_tamano(RotatedRect contorno,float razon,int min, int max,bool opt)

{

   float error=0.4;

   float rmin=razon*(1-error);

   float rmax=razon*(1+error);

   int area=contorno.size.height*contorno.size.width;

   float r=(float)contorno.size.width/(float)contorno.size.height;

   if(r<1)

      r=1/r;

   if(opt==1)

      cout<<r<<"     "<<area<<endl;

   if((area<min||area>max)||(r<rmin||r>rmax))

   {

      cout<<"FALSE"<<endl;

      return false;

   }
```

```
    else
    {
        cout<<"TRUE"<<endl;
        return true;
    }
}
vector<RotatedRect> f_contorno(Mat img,float razon,int min,int max,bool opt)
{
 //Funcion que detecta los contornos en la imagen
    Mat m_contorno=img;
    vector<vector<Point> > contorno;
    vector<RotatedRect> vec_rect;

findContours(m_contorno,contorno,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
    vector<vector<Point> >::iterator i=contorno.begin();
    while(i!=contorno.end())
    {
        RotatedRect rect=minAreaRect(Mat(*i));
        if(!verif_tamano(rect,razon,min,max,opt))
        {
            i=contorno.erase(i);
        }
        else
        {
            ++i;
            vec_rect.push_back(rect);
        }
    }
    return vec_rect;
}
Mat dibu_contorno(Mat img, vector<RotatedRect>vec_rect)
{
    Mat dibujo=img;
```

```
    Point2f vertices[4];
    for(unsigned i=0;i<vec_rect.size();i++)
    {
        RotatedRect m_rectangulo=vec_rect[i];
        m_rectangulo.points(vertices);
        for(int j=0;j<4;j++)
            line(dibujo,vertices[j],vertices[(j+1)%4],Scalar(255,0,0));
    }
    return dibujo;
}
int minimo(int a,int b,int c,int d)
{
    int aux[4]={a,b,c,d};
    int aux2=aux[0];
    for(int i=0;i<4;i++)
    {
        if(aux[i]<aux2)
        {
            aux2=aux[i];
        }
    }
    return aux2;
}
int maximo(int a,int b,int c,int d)
{
    int aux[4]={a,b,c,d};
    int aux2=aux[0];
    for(int i=0;i<4;i++)
    {
        if(aux[i]>aux2)
        {
            aux2=aux[i];
```

```
        }
    }
    return aux2;
}
int main()
{
    Mat img= imread("/home/tesis/medidores/datos/m54.jpg",0);
    Mat img2=imread("/home/tesis/medidores/datos/m54.jpg");
    resize(img,img,Size(round(img.cols/4),round(img.rows/4)));
    resize(img2,img2,Size(round(img2.cols/4),round(img2.rows/4)));
    blur( img2, img2, Size( 3, 3 ), Point(-1,-1) );
    imshow("Medidor Original",img2);
    Mat im1,im2,bin;
    threshold(img,bin,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY);
    Mat elemento1=getStructuringElement(MORPH_ELLIPSE, Size(3,3),Point(-1,-1)); //3,3
    dilate(bin,im1,elemento1);
    Mat elemento2=getStructuringElement(MORPH_RECT, Size(5,3),Point(-1,-1));    //5,3
    dilate(im1,im2,elemento2);
    Mat elemento3=getStructuringElement(MORPH_RECT, Size(15,5),Point(-1,-1));   //15,5
    erode(im2,im2,elemento3);
    threshold(im2,im2,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY_INV);
    imshow("Deteccion BLOBS",im2);
    vector<RotatedRect> vec_rect;
    float razon=5;
    int min=2500;
    int max=18000;
    vec_rect=f_contorno(im2,razon,min,max,1);
    if(vec_rect.size()>0)
    {
        RotatedRect rect=vec_rect[0];
        Mat M, rotated, cropped;
        float angle = rect.angle;
```

```
        Size rect_size = rect.size;
        if (rect.angle < -45.) {
            angle += 90.0;
            swap(rect_size.width, rect_size.height);
        }
        M = getRotationMatrix2D(rect.center, angle, 1.0);
        warpAffine(img, rotated, M, img.size(), INTER_CUBIC);
        getRectSubPix(rotated, rect_size, rect.center, cropped);
        Mat final=cropped;
        vector<int> compression_params;
        compression_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
        compression_params.push_back(9);
        imwrite("/home/tesis/medidores/recortes/recorte1.jpg",final,compression_params);
         imshow("Lectura Consumo",final);
    }
    cout<<endl<<"ok";
    while(1)
    {
        if(waitKey(30)==27)
            break;
    }
}
```

```
#include <iomanip>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <math.h>
#include <fstream>
#include <tesseract/baseapi.h>
#include <leptonica/allheaders.h>
#include <tesseract/strngs.h>

using namespace std;

using namespace cv;

Mat tratamiento_recorte(Mat img)

{

   Mat aux;

   Mat s1=getStructuringElement(MORPH_ELLIPSE, Size(6,6),Point(-1,-1));//(9,9)

   morphologyEx(img,aux,CV_MOP_TOPHAT,s1);

   threshold(aux,aux,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY);

   Mat s2=getStructuringElement(MORPH_ELLIPSE, Size(3,3),Point(-1,-1));

   dilate(aux,aux,s2);

   threshold(aux,aux,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY_INV);

   return aux;

}

int main()

{

  Mat img=imread("/home/tesis/medidores/recortes/recorte1.jpg",0);

   imshow("orig",img);

   GaussianBlur(img,img,Size( 3, 3 ), 0, 0 );

   GaussianBlur(img,img,Size( 3, 3 ), 0, 0 );


   imshow("Filtro Gauss",img);
```

```
    Mat final=tratamiento_recorte(img);
    vector<int> compression_params;
    compression_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
    compression_params.push_back(9);
    imwrite("/home/tesis/medidores/tratam/trat_rec1.jpg",final,compression_params);
    imshow("Binarizacion",final);
    while(1)
    {
       if(waitKey(0)==27)
          break;
    }
    return 1;
}
```

ANEXO # 2

```
// Programa que realiza el reconocimiento de los dígitos la imagen del medidor mediante RNA
// Autor: Janeth Arias G. 2016


#include <iomanip>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <math.h>
#include <fstream>
#include <ctime>
#include <iostream>
#define pres 200
#define num_caracteres 10
#define num_entradas 626
#define num_datos 10160
#define entrenar 1
#define neu_capa1 1000
#define neu_capa2 250
#define neu_capa3 10
#define l_imagen 25

using namespace std;

using namespace cv;

double pesos1[num_entradas][neu_capa1];
double pesos2[neu_capa1+1][neu_capa2];
double pesos3[neu_capa2+1][neu_capa3];


double salida1[neu_capa1+1];
double salida2[neu_capa2+1];
double salida3[neu_capa3+1];

int entrada[num_entradas];

Mat tratamiento_recorte(Mat img)

{

   Mat aux;

   Mat s1=getStructuringElement(MORPH_ELLIPSE, Size(9,9),Point(-1,-1));//(9,9)

   morphologyEx(img,aux,CV_MOP_TOPHAT,s1);

   threshold(aux,aux,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY);
```

```
    Mat s2=getStructuringElement(MORPH_ELLIPSE, Size(5,5),Point(-1,-1));
    threshold(aux,aux,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY);
    return aux;
}
string convert_char(int variable)
{
    stringstream numero;//crea un flujo de cadenas
    numero<<variable;//anade un numero a la cadena
    return numero.str();//return a string with the contents of the stream
}
void leer_pesos()
{
cout<<endl<<"Leyendo Pesos RNA";
  ifstream archivo1("pesos11.txt");
  for(int i=0;i<num_entradas;i++)
  {
  for(int j=0;j<neu_capa1;j++)
  {
  archivo1>>pesos1[i][j];
  }
  }
  archivo1.close();
   ifstream archivo2("pesos22.txt");
  for(int i=0;i<(neu_capa1+1);i++)
  {
  for(int j=0;j<neu_capa2;j++)
  {
  archivo2>>pesos2[i][j];
  }
  }
  archivo2.close();
```

```
    ifstream archivo3("pesos33.txt");
  for(int i=0;i<(neu_capa2+1);i++)
  for(int j=0;j<neu_capa3;j++)
  {
  archivo3>>pesos3[i][j];
  }
  }
  archivo3.close();
}
void init_pesos()
{
cout<<endl<<"iniciando pesos";
srand (time(NULL));
for(int i=0;i<neu_capa1;i++)
{
    for(int j=0;j<num_entradas;j++)
    pesos1[j][i]=1.0/(rand()%10+1)*(pow(-1,rand()%2));
}
for(int i=0;i<neu_capa2;i++)
{
    for(int j=0;j<(neu_capa1+1);j++)
    pesos2[j][i]=1.0/(rand()%10+1)*(pow(-1,rand()%2));
}
for(int i=0;i<neu_capa3;i++)
{
    for(int j=0;j<(neu_capa2+1);j++)//pesos por neurona/capa2+1bias
    pesos3[j][i]=1.0/(rand()%10+1)*(pow(-1,rand()%2));
}
salida1[neu_capa1]=1;
salida2[neu_capa2]=1;
}
void propagacion_adelante()
```

```
{
   double suma=0;
for(int i=0;i<neu_capa1;i++)
{
   suma=0;
   for(int j=0;j<num_entradas;j++)
   suma+=entrada[j]*pesos1[j][i];
   salida1[i]=tanh(suma);
}
for(int i=0;i<neu_capa2;i++)
{
   suma=0;
   for(int j=0;j<(neu_capa1+1);j++)
      suma+=salida1[j]*pesos2[j][i];
   salida2[i]=tanh(suma);
}
for(int i=0;i<neu_capa3;i++)
{
   suma=0;
   for(int j=0;j<(neu_capa2+1);j++)
      suma+=salida2[j]*pesos3[j][i];
   salida3[i]=tanh(suma);
   if(salida3[i]>=0.5)
      salida3[i]=1.0;
   if(salida3[i]<(-0.5))
      salida3[i]=-1.0;
}
}
bool verif_tamano_caracter(RotatedRect contorno)
{
   float error=0.5;
   const float razon=1.8;
```

```
    int min=200;
    int max=1000;
    float rmin=razon*(1-error);
    float rmax=razon*(1+error);
    int area=contorno.size.height*contorno.size.width;
    float r=(float)contorno.size.width/(float)contorno.size.height;
    if(r<1)
        r=1/r;
    cout<<r<<"  "<<area<<"  "<<endl;
    if((area<min||area>max))
    {
        cout<<"false"<<endl;
        return false;
    }
    else
    {
        cout<<"true"<<endl;
        return true;
    }
}
Mat dibu_contorno(Mat img, vector<RotatedRect>vec_rect)
{
    Mat dibujo=img;
    Point2f vertices[4];
    for(unsigned i=0;i<vec_rect.size();i++)
    {
        RotatedRect m_rectangulo=vec_rect[i];
        m_rectangulo.points(vertices);
        for(int j=0;j<4;j++)
            line(dibujo,vertices[j],vertices[(j+1)%4],Scalar(255,0,0));
    }
    return dibujo;
```

```
}
vector<RotatedRect> f_contorno(Mat &img)
{
   Mat m_contorno=img;
   vector<vector<Point> > contorno;
   vector<RotatedRect> vec_rect;

findContours(m_contorno,contorno,CV_RETR_EXTERNAL,CV_CHAIN_APPROX_NONE);
   vector<vector<Point> >::iterator i=contorno.begin();
   while(i!=contorno.end())
   {
      RotatedRect rect=minAreaRect(Mat(*i));
      if(!verif_tamano_caracter(rect))
      {
         i=contorno.erase(i);
      }
      else
      {
         ++i;
         vec_rect.push_back(rect);
      }
   }
   return vec_rect;
}
string clasificador()
{
   string caracter;
   int aux_valor=-2,aux_pos=0,aux_rep=0;
   for(int i=0;i<neu_capa3;i++)
   {
      if(salida3[i]>=aux_valor)
      {
         if(salida3[i]>aux_valor)
```

```
        {
            aux_valor=salida3[i];
            aux_pos=i;
            aux_rep=0;
        }
        else
        {
            aux_rep+=1;
        }
    }
}
if(1)
{
    switch(aux_pos)
    {
        case 0:caracter='0';break;
        case 1:caracter='1';break;
        case 2:caracter='2';break;
        case 3:caracter='3';break;
        case 4:caracter='4';break;
        case 5:caracter='5';break;
        case 6:caracter='6';break;
        case 7:caracter='7';break;
        case 8:caracter='8';break;
        case 9:caracter='9';break;
        default : caracter='#';
    }
}
else
{
    caracter="$";
}
```

```
    return caracter;
}
void generador(Mat &x)
{
int contador=0;
for(int i=0;i<x.rows;i++)
{
for(int j=0;j<x.cols;j++)
{
    if((x.at<uchar>(i,j))>127)
{
        entrada[contador]=-1;
}
    else
    {
        entrada[contador]=1;
    }
    contador+=1;
}
}
entrada[num_entradas-1]=1;
}
Mat f_binarizacion(Mat &img)
{
    Mat binaria;
    threshold(img,binaria,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY);
    return binaria;
}
Mat recortar(Mat &ej)
{
int xi=0;
int xd=0;
```

```
int yu=0;
int yd=0;
int encontrado=0;
Vec3b bgr;
for(int i=0;i<ej.rows;i++)
{
   if(encontrado==0)
   {
      for(int j=0;j<ej.cols;j++)
      {
         bgr=ej.at<Vec3b>(i,j);
         if((bgr[0]<30)&&(bgr[1]<30)&&(bgr[2]<30))
         {
            yu=i;
            encontrado=1;
         }
      }
   }
}
encontrado=0;
for(int i=ej.rows-1;i>=0;i--)
{
   if(encontrado==0)
   {
      for(int j=0;j<ej.cols;j++)
      {
         bgr=ej.at<Vec3b>(i,j);
         if((bgr[0]<30)&&(bgr[1]<30)&&(bgr[2]<30)) //selecciona color
         {
            yd=i;
            encontrado=1;
         }
```

```
        }
    }
}
encontrado=0;
for(int i=0;i<ej.cols;i++)
{
    if(encontrado==0)
    {
        for(int j=0;j<ej.rows;j++)
        {
            bgr=ej.at<Vec3b>(j,i);
            if((bgr[0]<30)&&(bgr[1]<30)&&(bgr[2]<30)) //selecciona color
            {
                xi=i;
                encontrado=1;
            }
        }
    }
}
encontrado=0;
for(int i=ej.cols-1;i>=0;i--)
{
    if(encontrado==0)
    {
        for(int j=0;j<ej.rows;j++)
        {
            bgr=ej.at<Vec3b>(j,i);
            if((bgr[0]<30)&&(bgr[1]<30)&&(bgr[2]<30))
            {
                xd=i;
                encontrado=1;
            }
```

```
        }


    }
}

Mat crop(ej,Rect(xi,yu,xd-xi,yd-yu));
return crop;
}
string clasificar_caracter(Mat &img,int i)
{
    Mat binaria,recortada,redimensionada;
    binaria=f_binarizacion(img);
    recortada=recortar(binaria);
    resize(recortada,redimensionada,Size(25,25));
    //resize(binaria,redimensionada,Size(25,25));
    imshow(convert_char(i),redimensionada);
    vector<int> compression_params;
    compression_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
    compression_params.push_back(9);

imwrite("/home/tesis/medidores/final/rec"+convert_char(i)+".png",redimensionada,compression
_params);




    generador(redimensionada);
    propagacion_adelante();
    return clasificador();
}
int minimo(int a,int b,int c,int d)
{
    int aux[4]={a,b,c,d};
    int aux2=aux[0];
```

```
    for(int i=0;i<4;i++)
    {
        if(aux[i]<aux2)
        {
            aux2=aux[i];
        }
    }
    return aux2;
}
int maximo(int a,int b,int c,int d)
{
    int aux[4]={a,b,c,d};
    int aux2=aux[0];
    for(int i=0;i<4;i++)
    {
        if(aux[i]>aux2)
        {
            aux2=aux[i];
        }
    }
    return aux2;
}
string obtener_string(Mat img,vector<RotatedRect>vec_rect)
{
    cout<<endl<<"Obtener string   "<<vec_rect.size()<<endl;
    string cadena=" ";
    Mat dibujo=img;
    Point2f vertices[4];
    for(int i=vec_rect.size()-1;i>=0;i--)
    {
        RotatedRect m_rectangulo=vec_rect[i];
        m_rectangulo.points(vertices);
```

```
        int x1=minimo(int(vertices[1].x),int(vertices[2].x),int(vertices[3].x),int(vertices[0].x));
        int y1=minimo(int(vertices[1].y),int(vertices[2].y),int(vertices[3].y),int(vertices[0].y));
        int x2=maximo(int(vertices[1].x),int(vertices[2].x),int(vertices[3].x),int(vertices[0].x));
        int y2=maximo(int(vertices[1].y),int(vertices[2].y),int(vertices[3].y),int(vertices[0].y));


        if((x2-x1)>0&&(y2-y1)>0)
        {
           cout<<x2-x1<<"  "<<y2-1<<endl;
          cout<<"cumple:
("<<int(vertices[1].x)<<","<<int(vertices[1].y)<<")("<<int(vertices[2].x)<<","<<int(vertices[2].y)<<")("<
<int(vertices[3].x)<<","<<int(vertices[3].y)<<")("<<int(vertices[0].x)<<","<<int(vertices[0].y)<<")"<<en
dl;
           Mat recorte;
           img(Rect(x1,y1,x2-x1,y2-y1)).copyTo(recorte);
           string aux=clasificar_caracter(recorte,i);
           cadena=cadena+aux;
           imshow(convert_char(i),recorte);
        }
        else
        {
          cout<<"NO cumple:
("<<int(vertices[1].x)<<","<<int(vertices[1].y)<<")("<<int(vertices[2].x)<<","<<int(vertices[2].
y)<<")("<<int(vertices[3].x)<<","<<int(vertices[3].y)<<")("<<int(vertices[0].x)<<","<<int(verti
ces[0].y)<<")"<<endl;
        }
        //cout<<(vertices[0])<<" "<<(vertices[1])<<(vertices[2])<<" "<<(vertices[3])<<endl;
        //Mat recorte(dibujo,Rect(200,200,50,50));
    }
    cout << "Resultado:" << endl;
    cout << cadena;
    return cadena;
}
string ocr(Mat img,Mat img2)
{
    vector<RotatedRect> vec_rect;
```

```
    vec_rect=f_contorno(img);    return obtener_string(img2,vec_rect);
    return "OK";
}
int main()
{
    unsigned t0,t1;
    t0=clock();
    init_pesos();
    leer_pesos();
    cout<<"Pesos:  "<<pesos2[0][0]<<endl;
    String dir ="/home/tesis/medidores/tratam/trat_rec1.jpg";
    String dir2="/home/tesis/medidores/recortes/recorte1.jpg";
    Mat img= imread(dir,0);
    Mat img2=imread(dir,0);
    imshow("recorte",img);
    Mat img3;
    threshold(img,img3,0,255,CV_THRESH_OTSU+CV_THRESH_BINARY_INV);
    cout<<ocr(img3,img2);   //ocr llama al programa de red neuronal
    cout<<endl<<endl<<" ok fin" << endl;
    t1=clock();
    double time=(double(t1-t0)/CLOCKS_PER_SEC);
    cout << "Tiempo: " << time << endl;
    while(1)
    {
       if(waitKey(0)==27)
           break;
    }
    return 1;
}
```

ANEXO # 3

```
// Entrenamiento de Datos mediante KNN
// Autor: Janeth Arias G. 2016

import sys

import numpy as np

import cv2

import os

MIN_CONTOUR_AREA = 100

RESIZED_IMAGE_WIDTH = 20

RESIZED_IMAGE_HEIGHT = 30

def main():

    imgTrainingNumbers = cv2.imread("traintes.png")

    if imgTrainingNumbers is None:

        print "error: La imagen no puede ser leida \n\n"

        os.system("pause")

        return

    # end if

    imgGray = cv2.cvtColor(imgTrainingNumbers, cv2.COLOR_BGR2GRAY)
   imgBlurred = cv2.GaussianBlur(imgGray, (5,5), 0)
   imgThresh = cv2.adaptiveThreshold(imgBlurred,  255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,   cv2.THRESH_BINARY_INV,   11,  2)
#

    cv2.imshow("imgThresh", imgThresh)
   imgThreshCopy = imgThresh.copy()
   imgContours, npaHierarchy = cv2.findContours(imgThreshCopy
cv2.RETR_EXTERNAL,   cv2.CHAIN_APPROX_SIMPLE)

 npaFlattenedImages =  np.empty((0, RESIZED_IMAGE_WIDTH *
RESIZED_IMAGE_HEIGHT))

intClassifications = []

intValidChars = [ord('0'), ord('1'), ord('2'), ord('3'), ord('4'), ord('5'), ord('6'), ord('7'), ord('8'),
ord('9'), ord('A'), ord('B'), ord('C'), ord('D'), ord('E'), ord('F'), ord('G'), ord('H'), ord('I'), ord('J'),

 ord('K'), ord('L'), ord('M'), ord('N'), ord('O'), ord('P'), ord('Q'), ord('R'), ord('S'), ord('T'),

 ord('U'), ord('V'), ord('W'), ord('X'), ord('Y'), ord('Z')]

npaContours = imgContours

for npaContour in npaContours:

if cv2.contourArea(npaContour) > MIN_CONTOUR_AREA:
```

```
        [intX, intY, intW, intH] = cv2.boundingRect(npaContour

        cv2.rectangle(imgTrainingNumbers,

                (intX, intY),

                (intX+intW,intY+intH),
                (0, 0, 255),

                2)

        imgROI = imgThresh[intY:intY+intH, intX:intX+intW

        imgROIResized = cv2.resize(imgROI, (RESIZED_IMAGE_WIDTH,
RESIZED_IMAGE_HEIGHT))

        cv2.imshow("imgROI", imgROI)

        cv2.imshow("imgROIResized", imgROIResized

        cv2.imshow("Imagen_ Entrenamiento", imgTrainingNumbers)

        intChar = cv2.waitKey(0)

        if intChar == 27:

           sys.exit()

        elif intChar in intValidChars:

. .

intClassifications.append(intChar

npaFlattenedImage = imgROIResized.reshape((1, RESIZED_IMAGE_WIDTH *
RESIZED_IMAGE_HEIGHT))
npaFlattenedImages = np.append(npaFlattenedImages, npaFlattenedImage, 0)
# end if

      # end if

   # end for

fltClassifications = np.array(intClassifications, np.float32

npaClassifications = fltClassifications.reshape((fltClassifications.size, 1))

   print "\n\ntraining complete !!\n"

   np.savetxt("clasificaciones.txt", npaClassifications)

   np.savetxt("imagenes_planas.txt", npaFlattenedImages)

   cv2.destroyAllWindows()

   return

if __name__ == "__main__":

   main()

# end if
```

ANEXO # 4

```
// Programa que realiza el reconocimiento de dígitos mediante KNN
// Autor: Janeth Arias G. 2016
import cv2
import numpy as np
import operator
import os
MIN_CONTOUR_AREA = 100
RESIZED_IMAGE_WIDTH = 20
RESIZED_IMAGE_HEIGHT = 30
class ContourWithData():
    npaContour = None
    boundingRect = None
    intRectX = 0
    intRectY = 0
    intRectWidth = 0
    intRectHeight = 0
    fltArea = 0.0
 def calculateRectTopLeftPointAndWidthAndHeight(self):
        [intX, intY, intWidth, intHeight] = self.boundingRect
        self.intRectX = intX
        self.intRectY = intY
        self.intRectWidth = intWidth
        self.intRectHeight = intHeight
def checkIfContourIsValid(self):
        if self.fltArea < MIN_CONTOUR_AREA: return
        return True
def main():
    allContoursWithData = []
    validContoursWithData = []
    try:
        npaClassifications = np.loadtxt("classifications.txt", np.float32)
    except:
        print "error, No es posible abrir classifications.txt, saliendo del programa\n"
```

```
        os.system("pause")
        return
    # end try
    try:
        npaFlattenedImages = np.loadtxt("flattened_images.txt", np.float32)
    except:
        print "error, No es posible abrir flattened_images.txt, saliendo del programa\n"
        os.system("pause")
        return
    # end try
npaClassifications = npaClassifications.reshape((npaClassifications.size, 1))
    kNearest = cv2.KNearest()
    kNearest.train(npaFlattenedImages, npaClassifications)
    imgTestingNumbers = cv2.imread("m1f.png")
    if imgTestingNumbers is None:
        print "error: La imagen no se puede leer \n\n"
        os.system("pause")
        return
    # end if
imgGray = cv2.cvtColor(imgTestingNumbers, cv2.COLOR_BGR2GRAY)
    imgBlurred = cv2.GaussianBlur(imgGray, (5,5), 0)
    imgThresh = cv2.adaptiveThreshold(imgBlurred,
                                      255,
                                      cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY_INV,
                                      11,
                                      2)
    imgThreshCopy = imgThresh.copy()
    imgContours, npaHierarchy = cv2.findContours(imgThreshCopy,
                                         cv2.RETR_EXTERNAL,
                                         cv2.CHAIN_APPROX_SIMPLE)
    npaContours = imgContours
```

```
    for npaContour in npaContours:
        contourWithData = ContourWithData()
        contourWithData.npaContour = npaContour
        contourWithData.boundingRect = cv2.boundingRect(contourWithData.npaContour)
        contourWithData.calculateRectTopLeftPointAndWidthAndHeight()
        contourWithData.fltArea = cv2.contourArea(contourWithData.npaContour)
        allContoursWithData.append(contourWithData)
    # end for
    for contourWithData in allContoursWithData:
        if contourWithData.checkIfContourIsValid():
            validContoursWithData.append(contourWithData)
        # end if
    # end for

    validContoursWithData.sort(key = operator.attrgetter("intRectX"))
    strFinalString = ""
    for contourWithData in validContoursWithData:
            cv2.rectangle(imgTestingNumbers,
                (contourWithData.intRectX, contourWithData.intRectY),
                (contourWithData.intRectX + contourWithData.intRectWidth,
contourWithData.intRectY + contourWithData.intRectHeight),
                (0, 255, 0),
                2)
        imgROI = imgThresh[contourWithData.intRectY : contourWithData.intRectY +
contourWithData.intRectHeight,
                    contourWithData.intRectX : contourWithData.intRectX +
contourWithData.intRectWidth]
        imgROIResized = cv2.resize(imgROI, (RESIZED_IMAGE_WIDTH,
RESIZED_IMAGE_HEIGHT))
        npaROIResized = imgROIResized.reshape((1, RESIZED_IMAGE_WIDTH *
RESIZED_IMAGE_HEIGHT))

        npaROIResized = np.float32(npaROIResized)
        retval, npaResults, neigh_resp, dists = kNearest.find_nearest(npaROIResized, k = 1)
```

```
        strCurrentChar = str(chr(int(npaResults[0][0])))
        strFinalString = strFinalString + strCurrentChar
    print "\n NUMERO RECONOCIDO: "
    print "\n" + strFinalString + "\n"
    cv2.imshow("Consumo_Electrico", imgTestingNumbers)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return
if __name__ == "__main__":
    main()
# end if
```

ANEXO # 5

```
// Programa que realiza el reconocimiento de dígitos mediante Tessract
// Autor: Janeth Arias G. 2016

#include <tesseract/baseapi.h>
#include <leptonica/allheaders.h>

int main()
{
   char *outText;
   tesseract::TessBaseAPI *api = new tesseract::TessBaseAPI();
   // Inicializa tesseract con los datos TESIS obtenidos del entrenamiento
   if (api->Init(NULL, "tesis")) {
      fprintf(stderr, "Error: No puede inicializarse TESSERACT.\n");
      exit(1);
   }
   // Se abre la imagen a reconocer mediante la librería leptónica
   // Las imágenes se encuentran almacenadas en el directorio indicado
   Pix *image = pixRead("/home/tesis/medidores/datosf/m1f.png");
   api->SetImage(image);
   // Resultado del Reconocimiento
   outText = api->GetUTF8Text();
   printf("Lectura del Medidor:\n%s", outText);
   // Se destruye el objeto usado y se libera la memoria
   api->End();
   delete [] outText;
   pixDestroy(&image);
   return 0;
}
```