IMPLEMENTACIÓN DE UNA RED NEURONAL CONVOLUCIONAL PARA LA DETECCIÓN DE 3 OBJETOS DISTINTOS

Everardo Hernández Manuel[1], Jorge Rivero Gomez[1], Josue Roberto Rodríguez Cerón[1], Erik Reyes Reyes[2], Ramón Silva Ortigoza[2], Magdalena Marciano Melchor[2]

Instituto Politécnico Nacional

[1]Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA)
[2]CIDETEC, Laboratorio de Mecatrónica y Energía Renovable

ehernandezm1614@alumno.ipn.mx

Boletín No. 107, 1o. de marzo de 2025

Resumen

Se presenta la programación e implementación de una red neuronal convolucional en Python para la detección de objetos de 3 diferentes categorías, se muestra la construcción del conjunto de datos de entrenamiento, el código de programación de la red neuronal y el código de prueba de la red, el reconocimiento de objetos con este tipo de redes puede aplicarse a sistemas de visión artificial en diversos proyectos mecatrónicos.

Palabras Clave: redes neuronales convolucionales, visión artificial, detección de objetos, Python, aprendizaje profundo.

1. Introducción

Las redes neuronales convolucionales generalmente se construyen por 3 capas, la capa de convolución, la capa de reducción y las capas densas, las primeras 2 capas tienen la función de extraer las características, (color, bordes, curvas), la última capa se encarga de la clasificación, que detecta al objeto que se quiera identificar [1].

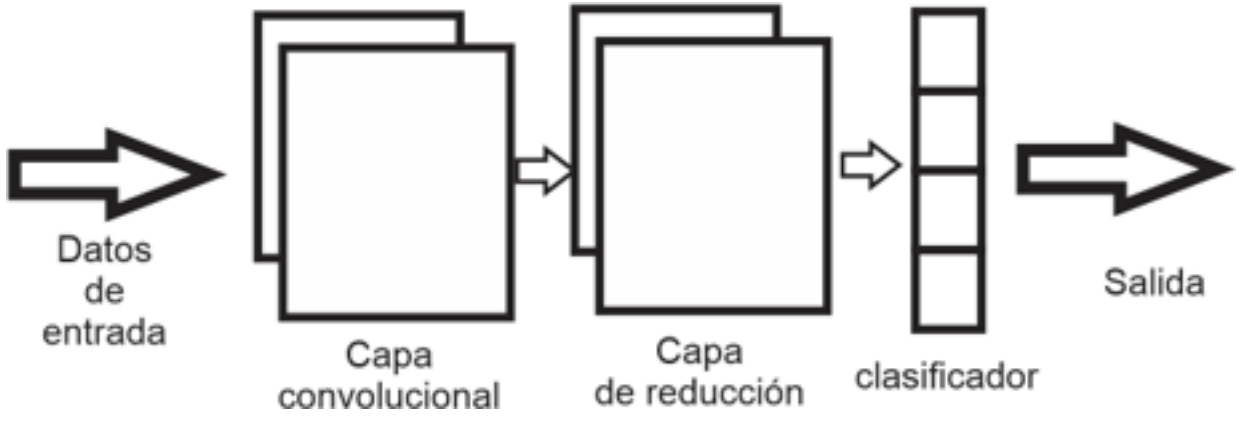


Figura 1 Estructura de una red neuronal convolucional.

La convolución aplica un filtro con un tamaño de ventana específico para de extraer ciertas características, este filtro se desplaza hasta recorrer todos los pixeles de la imagen, el resultado de aplicar el filtro es una matriz con las características detectadas de la imagen. Las capas de reducción (max pooling) resumen la información de salida de las capas de convolución, el proceso se repite hasta que se llega a la capa de clasificación.

2. Materiales y software utilizados

Para la implementación de la red neuronal convolucional se necesitan tener los siguientes programas y extensiones instalados:

- Computadora o laptop.
- Celular con CamoStudio instalado.
- Visual Studio Code.
- Extensión de Python para Visual Studio.
- Biblioteca tensorflow.
- Biblioteca matplotlib.
- Biblioteca OpenCV.
- Python versión 3.10.0.
- CamoStudio.
- Extensión “download all images” Google Chrome.
- Motores de CC, circuitos integrados y pinzas.

Requerimos tener instalado Python (versión 3.10.0 en este caso), Visual Studio Code, (donde se programa la red neuronal convolucional) con la extensión de Python, así como las librerías de tensorflow, matplotlib y OpenCV; la extensión “download all images” nos será de ayuda para crear los datos de entrenamiento; también se requiere CamoStudio en la computadora y en el celular.

El sistema en el que se va a probar la red neuronal se visualiza en la siguiente figura. Este consta de una computadora en la cual se ejecutará el código de reconocimiento de objetos, un celular que se configurará como cámara externa a través de CamoStudio, la cual tomará la imagen del objeto a reconocer.

Figura 2 Escenario de pruebas del objeto a reconocer.

3. Programación y comprobación de la red

3.1 Creación de los datos de entrenamiento y prueba

Proponemos 4 categorías de datos de entrenamiento, cada una con 300 imágenes, las primeras 3 categorías son los objetos que queremos detectar y la cuarta es el fondo. Por un lado, las categorías 1, 2 y 3 se construyeron a través de la recolección de imágenes con la extensión “download all images” de Google Chrome. Por otro lado, la categoría 4 se creó con fotografías propias. De esta manera, de las imágenes descargadas se seleccionaron aquellas que más se parecen al escenario de prueba y se agregaron fotos reales de los objetos para mejorar la calidad del entrenamiento. Parte de las imágenes de los datos de entrenamiento, así como el orden de las categorías se ve en las siguientes figuras:

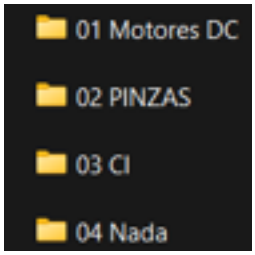
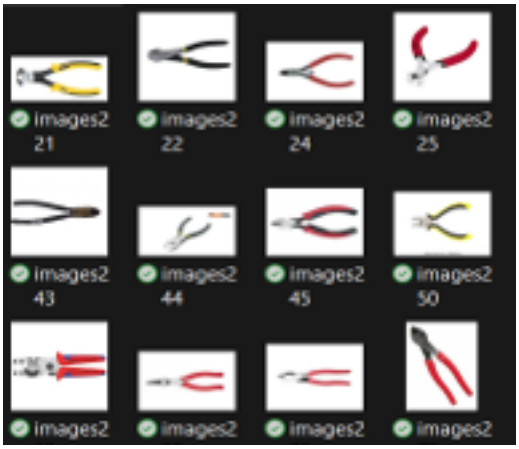
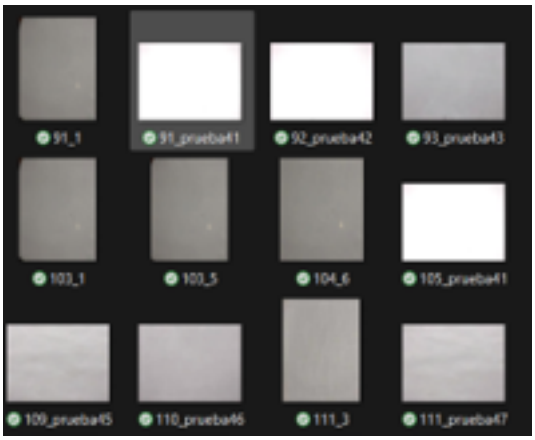


Figura 3 Carpetas con el nombre de objetos a clasificar.

a) Circuitos integrados.

c) Motores eléctricos.





d) Nada.

Figura 4 Ejemplos de las categorías de imágenes utilizadas para el entrenamiento.

3.2 Código de entrenamiento

El código de entrenamiento crea, entrena y descarga el modelo de la neurona que después será cargado a un programa de verificación.

Primero se importan las librerías que nos permiten graficar, generar más imágenes y construir la neurona, luego se crea el conjunto de datos de entrenamiento y validación, especificando el tamaño de las imágenes, el tamaño de lotes, la forma de clasificación y la etiqueta correspondiente.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
#GENERA DATASET
datagen = ImageDataGenerator(
    rescale = 1/255.0, #NORMALIZA
    rotation_range = 10,
    width_shift_range = 0.15,
    height_shift_range = 0.15,
    shear_range = 5,
    zoom_range = [0.7,1.3],
    validation_split = 0.2  #20% PARA VALIDACION
)
data_gen_entrenamiento = datagen.flow_from_directory(
    "Datasets2", #ruta
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'categorical', #categorical para softmax
    subset = "training"
)
data_gen_pruebas = datagen.flow_from_directory(
    "Datasets2", #ruta
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'categorical',
    subset = "validation"
)
```

Figura 5 Generación de datos de entrenamiento.

Luego se genera la arquitectura de la red neuronal. Para esto, utilizamos 4 filtros convolucionales, usamos funciones de activación tipo relu y al final de la neurona colocamos un clasificador tipo softmax, Posteriormente compilamos el modelo, lo entrenamos con 50 épocas y lo guardamos con extensión .h5.

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

model=Sequential()
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(224,224,3)))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
#model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512,activation='relu'))
model.add(Dense(4,activation='softmax'))

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

ENTRENAMIENTO = model.fit(data_gen_entrenamiento,epochs=50,batch_size=32)

model.save("ModEnt.h5")
```

Figura 6 Arquitectura, compilado, entrenamiento y guardado de la red neuronal convolucional.

Agregando las siguientes líneas de código para evaluarlas con las imágenes de prueba obtenemos una precisión de 95 %, el cual es un valor muy bueno de exactitud.

```python
test_loss,test_acc = model.evaluate(data_gen_pruebas)
print('Test loss: ',test_loss)
print('Test acc: ',test_acc)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 5s 719ms/step - accuracy: 0.9568 - loss: 0.1781
Test loss:  0.17443296313285828
Test acc:  0.9575471878051758
```

Figura 7 Precisión y pérdida con las imágenes de prueba.

3.3 Código de prueba

En este apartado se muestra el código de comprobación, que consiste en cargar el modelo salvado. También, se captura la imagen obtenida por una cámara de celular la cual se seleccionó mediante Camo Studio. Luego, se preparan las imágenes para evaluarlas y cuyo resultado corresponde con el índice de la clase que se parece más al objeto de prueba. Por último, se muestra la imagen capturada con el título del objeto identificado. Para detener el programa basta con presionar la tecla “ESC” (ver [2]).

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
clases = ["Motor","pinzas","CI","Nada"]
from keras.models import load_model
#model = load_model('modelo-ent-prueba1.h5')
model = load_model('ModEnt.h5')
import cv2
cap = cv2.VideoCapture(1)
import time
while cap.isOpened():
    ret,im = cap.read()

    if ret == False:
        break

    imagen = cv2.resize(im,(224,224))
    imagen = imagen/255.0
    prediccion = np.argmax(model.predict(imagen[None,...])) #quita capas
    texto = clases[prediccion]
    cv2.putText(im,texto,(100,100),1,3,(0,255,255),3)
    cv2.imshow("Practica 6",im)
    if cv2.waitKey(1) == 27:
        cv2.destroyAllWindows()
        break
```

Figura 8 Código de evaluación del funcionamiento de la red.

3.4 Prueba con detección de objetos

1. Configuramos el dispositivo como cámara externa.

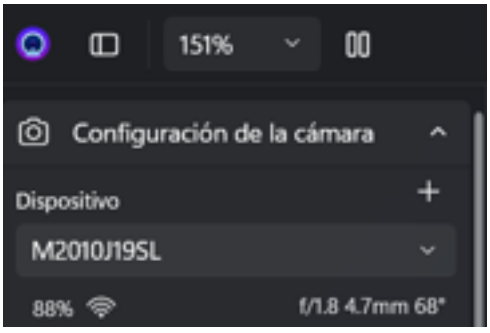
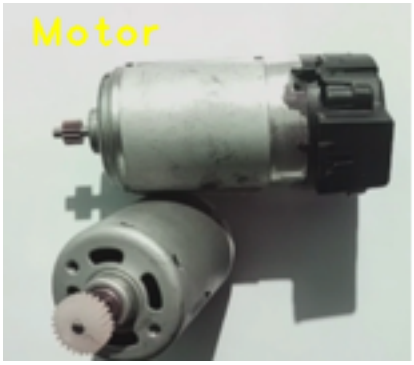


Figura 9 Configuración de CamoStudio con la cámara del celular.

2. Ejecutamos el código de comprobación y colocamos los objetos dentro del rango de visión de la cámara para que nos muestre su predicción. Los resultados se muestran a continuación:
   - Detección de motores de corriente directa:



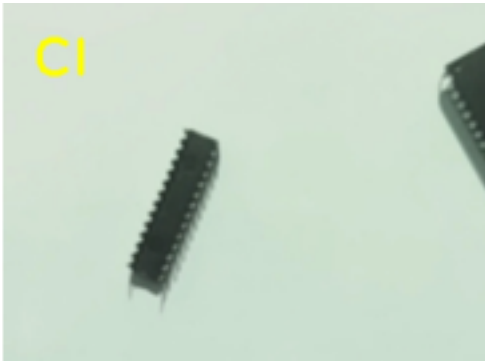


Figura 10 Detección de motores, pinzas y circuitos integrados.

- No detección de objetos:

Figura 11 Sin detección del objeto.

4. Conclusiones

La implementación de redes neuronales convolucionales en sistemas de visión artificial resulta ser una buena alternativa para la distinción de varios objetos pertenecientes a diferentes categorías en entornos controlados. En estas condiciones la variación de posición del objeto dentro de la captura de imagen resulta invariante ante la detección de los objetos.

Referencias

[1] Marconi, B., Rueda, C., Lubinus, F., Arías, Y. (**2024, Noviembre 28**). *Redes neuronales convolucionales: un modelo de Deep Learning en imágenes diagnósticas.* Revisión de tema. [Online]. Disponible en: `http://contenido.acronline.org/Publicaciones/RCR/RCR32-3/RCR%2032-3-03_Redes%20neuronales.pdf`

[2] s.a. (**s.f.**). *Camo - Usa tu teléfono como una cámara web profesional, gratis.* Reincubate — Get more from your devices. Accedido el 28 de noviembre de 2024. [En línea]. Disponible en: `https://reincubate.com/es/camo/use-phone-as-webcam/`

Hernández Manuel, E., Rivero Gomez, J., Rodríguez Cerón, J. R., Reyes Reyes, E., Silva Ortigoza, R., Marciano Melchor, M. *(2026). IMPLEMENTACIÓN DE UNA RED NEURONAL CONVOLUCIONAL PARA LA DETECCIÓN DE 3 OBJETOS DISTINTOS. Boletín UPIITA. año XX, (NÚM) 2026.*