PROTOTIPO FUNCIONAL PARA CLASIFICACIÓN DE IMÁGENES CON SALIDA DE AUDIO EN UN SISTEMA EMBEBIDO CON RED NEURONAL CONVOLUCIONAL

FUNCTIONAL PROTOTYPE FOR CLASSIFICATION OF IMAGES WITH AUDIO OUTPUT IN AN EMBEDDED SYSTEM USING CONVOLUTIONAL NEURAL NETWORK

Fidel López Saca
Universidad Autónoma Metropolitana
fidelosmcc@gmail.com

Andrés Ferreyra Ramírez
Universidad Autónoma Metropolitana
fra@correo.azc.uam.mx

Carlos Avilés Cruz
Universidad Autónoma Metropolitana
caviles@correo.azc.uam.mx

Resumen

En los últimos años, las redes neuronales convolucionales, han tenido una gran popularidad en aplicaciones de clasificación de imágenes, principalmente porque superan en rendimiento a los algoritmos tradicionales. Sin embargo, su alto costo computacional complica su implementación en sistemas embebidos con pocos recursos como las Raspberry Pi 3. Para superar este problema, se puede hacer uso del “Neural Compute Stick”, un dispositivo desarrollado recientemente, que integra una GPU en la que se puede cargar una red neuronal convolucional pre-entrenada. En este artículo se presenta un prototipo basado en la Raspberry Pi 3, que realiza clasificación de imágenes con reproducción de audio. La clasificación se realizada con la red GoogleNet, la cual es entrenada fuera de línea, implementada en un NCS e integrada a la tarjeta Raspberry Pi 3. En el sistema propuesto, la imagen que ingresa a través de una cámara web, es clasificada y etiquetada con la red convolucional y finalmente la etiqueta es traducida en audio por el sistema embebido para describir el objeto encontrado en la imagen.

Palabras Claves: Movidius stick, Raspberry, red neuronal convolucional, TensorFlow, visión por computador.

Abstract

In recent years, convolutional neural networks (CNN) have become very popular in image classification applications, mainly because they outperform traditional algorithms in performance. However, its high computational cost complicates its implementation in embedded systems with few resources such as Raspberry Pi 3. To overcome this problem, a "Neural Compute Stick" (NCS) can be used, which integrates a GPU. In the NCS can be loaded a pre-trained convolutional neural network. This article presents a prototype based on a Raspberry Pi 3, which performs image classification with audio reproduction. The classification is done through a GoogleNet net, which is trained offline, implemented in the NCS and integrated with the Raspberry Pi 3 card. In the proposed system, the image that enters through a webcam is classified and tagged with the CNN. Finally, the tag is translated into an audio file to be heard.

Keywords: Computer vision, movidius stick, Raspberry, convolutional neural network, TensorFlow.

1. Introducción

El aprendizaje profundo es un subcampo específico del aprendizaje automático, un enfoque que utiliza arquitecturas compuestas de múltiples capas de transformaciones no lineales, para aprender representaciones a partir de datos y que pone énfasis en el aprendizaje de capas sucesivas de representaciones cada vez más significativas. Una de las arquitecturas más populares utilizadas hoy en día son las redes neuronales convolucionales (CNN, por sus siglas en inglés), ya que cada una de sus capas, modela un campo receptivo de la corteza visual primaria del cerebro biológico; lo que las hace muy efectivas en tareas de visión artificial. Dado que las CNN emulan la visión humana, tienen una alta precisión para tareas de reconocimiento de imágenes y han demostrado un rendimiento casi humano en muchas tareas de visión artificial del mundo real, incluyendo la clasificación de

imágenes [Ciresan, 2011] [Krizhevsky, 2012], el reconocimiento [Zhang, 2015] [Mollahosseini, 2016] y la detección de objetos [Sermanet, 2014] [Girshick, 2014], el diagnóstico por imágenes médicas [Prasoon, 2013], la segmentación de escenas y el etiquetado [Karpathy, 2015], etc.

El entrenamiento de una CNN compleja representa un alto costo computacional, por lo que, la mayoría de modelos están entrenados e implementados con plataformas de software, utilizando potentes unidades de procesamiento gráfico (GPU, por sus siglas en inglés) [Vasilache, 2014] [Strigl, 2010] como motores de cálculo, que tienen numerosas unidades de ejecución y gran ancho de banda de memoria para obtener un alto rendimiento computacional.

En aplicaciones de reconocimiento en tiempo real, las redes pequeñas (con pocas capas) han sido implementadas en dispositivos móviles con pocos recursos de memoria y procesamiento, por ejemplo: MobileNet [Howard, 2017] entrenada con ImageNet [Stanford, 2017] para la clasificación de 1000 objetos diferentes, se ha implementado en celulares con sistema operativo Android para detectar la retinopatía diabética, entrenada y probada con más de 16,000 imágenes preprocesadas para la detección de este problema [Suriyal, 2018]. Sin embargo, redes muy grandes (con muchas capas) no pueden ser implementadas en dispositivos móviles, ni en sistemas embebidos de bajo costo tales como: Arduino, Beagle y Raspberry, ya que sus procesadores de propósito general no han sido optimizados para la implementación de estas redes.

Las CNN han sido implementadas en sistemas de visión embebidos para aplicaciones tales como: reconocimiento de escritura a mano [LeCun, 1998], detección de rostros [Sankaradas, 2009], reconocimiento de objetos [Farabet, 2010], y determinación de escenas [Peemen, 2013]. En estos sistemas de visión embebidos, el entrenamiento de la CNN se realizó fuera de línea, y el modelo pre-entrenado se ejecutó en tiempo real. Utilizaron procesadores demasiado lentos lo que demandaba la aceleración de la CNN para la ejecución en tiempo real, la cual solo se podía lograr con la integración de GPU en los sistemas embebidos; sin embargo, la solución no era muy adecuada ya que las GPU consumían muy alta potencia.

A principios del 2014, empezaron a surgir supercomputadoras móviles para el desarrollo de sistemas embebidos las cuales aprovechan los recursos de cómputo de procesadores de primera generación (con diferentes arquitecturas y GPU) que posibilitan el desarrollo de una nueva generación de aplicaciones que emplean la visualización por computadora, el procesamiento de imágenes y el procesamiento de datos en tiempo real, para los sectores de robótica, medicina, aeroespacial y fabricación de automóviles. Estas plataformas año con año duplican su rendimiento y reducen el consumo de potencia, lo que las hace muy atractivas para entrenar e implementar CNN embebidas; sin embargo, tienen un problema, son relativamente costosas y de difícil acceso.

Los problemas de implementación de CNN en sistemas embebidos de bajo costo como la Raspberry Pi 3, pueden ser resueltos con el uso de un Neural Compute Stick (NCS) [Xu, 2017], un dispositivo con conexión USB de bajo consumo de potencia y costo, que integra una GPU que soporta la carga de redes diseñadas y entrenadas en plataformas de aprendizaje profundo como Caffe [Caffe, 2018] y TensorFlow [TensorFlow, 2018]; éste dispositivo soporta redes tales como: AlexNet [Krizhevsky, 2012], GoogleNet [Szegedy, 2015] y ResNet [He, 2016].

La implementación de CNN en sistemas embebidos de bajo costo, puede acelerar el camino de la computación embebida hacia el futuro, permitiendo el desarrollo de aplicaciones en donde las máquinas interactúen y se adapten a sus entornos en tiempo real. En este artículo se presenta un prototipo basado en la Raspberry Pi 3, que realiza clasificación de imágenes con reproducción de audio. La clasificación es realizada por la CNN GoogleNet, la cual es entrenada fuera de línea, implementada en un NCS e integrada a la tarjeta Raspberry Pi 3.

2. Métodos

El prototipo está dividido en dos etapas: la etapa de entrenamiento y evaluación de la CNN y la etapa de implementación de la red en el sistema embebido Raspberry Pi 3, figura 1. En esta sección se describe: cada uno de los dispositivos principales del prototipo, el entrenamiento y la evaluación de la CNN, la implementación de la CNN en el NCS, y el conjunto de datos utilizado.

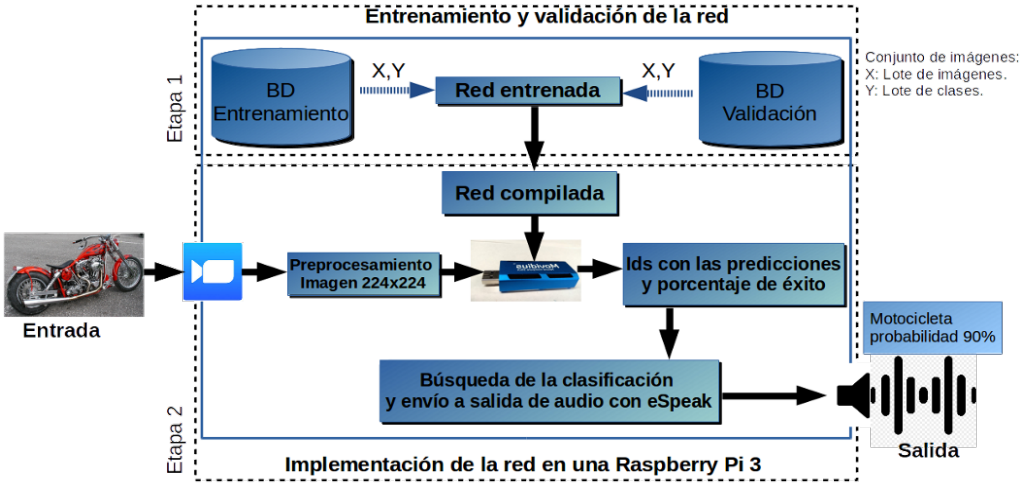

Figura 1 Metodología general.

Raspberry Pi 3

La Raspberry Pi 3 [Raspberry Pi, 2018], es una poderosa mini computadora basada en un procesador ARMv8 Cortex A53, un procesador de cuatro núcleos a 1.2 GHz de 64 bits. Está equipada con: 1 GB de memoria RAM, 802.11n Wireless LAN (10/100Mbps de velocidad), Bluetooth 4.1 de baja energía (BLE), 4 puertos USB, 40 pines GPIO (General Purpose Input Output), puerto HDMI completo, Interfaz de cámara (CSI), Interfaz de pantalla (DSI), ranura para tarjeta Micro SD, salida estéreo y un procesador gráfico VideoCore IV 3D, ver figura 2. Su precio asequible, compatibilidad con otros dispositivos, alta flexibilidad, rendimiento adecuado, recursos disponibles y una amplia comunidad de usuarios; hacen de esta plataforma, la elección ideal para desarrollar aplicaciones embebidas basadas en aprendizaje automático.

Neural Compute Stick (NCS)

El Intel® Movidius™ Neural Compute Stick, es un pequeño dispositivo USB 3.0 de bajo costo diseñado para implementar redes de aprendizaje profundo, en especial CNN, en aplicaciones de baja potencia que requieren inferencia en tiempo real. Internamente contiene un procesador Movidius™ Myriad™ 2, una unidad de procesamiento de visión (VPU, por sus siglas en inglés) que ofrece un alto rendimiento en entornos con energía limitada y su arquitectura no depende de una

conexión a Internet. Está diseñado para trabajar en un rango de temperatura de 0 a 40 °C y sus dimensiones son de 72.5mm x 14mm, figura 2. En [Xu, 2017] se muestran comparaciones de su capacidad de procesamiento.

El NCS tiene un toolkit “Intel® Movidius™ NCS Quick Start” [NCS Quick Star, 2018], y ejemplos “Neural Compute Application Zoo (NC App Zoo)” [Neural, 2018] con Caffe [Caffe, 2018] y TensorFlow [TensorFlow, 2018].

Figura 2 Dispositivos principales del prototipo.

Red neuronal convolucional: entrenamiento y prueba

La CNN adoptada para el sistema propuesto es GoogleNet [Szegedy, 2015], un modelo de 144 capas que se puede dividir en diferentes secciones básicas: el tallo (stem), los módulos de inicio (inception modules), clasificadores auxiliares, y el clasificador de salida. El tallo está formado por una secuencia de operaciones de convolución, agrupación (pooling) y operaciones de normalización de respuesta local. Cada uno de los nueve módulos de inicio, está formado por un conjunto de convoluciones y agrupaciones a diferentes escalas, cada una hecha en paralelo y posteriormente concatenadas. Los clasificadores auxiliares se ramifican desde la red principal, y tienen como objetivo amplificar la señal de gradiente a través de la red, intentando mejorar las representaciones anteriores de los datos. Finalmente, el clasificador de salida, realiza una operación de agrupación promedio (average pooling) seguida de una activación “softmax” en una capa totalmente conectada.

La CNN fue entrenada utilizando descenso de gradiente estocástico con un tamaño de lote de 32, momento de 0.9 y una disminución de pesos (factor de regularización) de 0.00004. Los pesos iniciales en cada una de las capas de convolución y completamente conectas, fueron inicializados utilizando el algoritmo de inicialización Xavier; en el cual, los pesos son aleatoriamente inicializados con una

media cero y una varianza que depende de la cantidad de conexiones entrantes y salientes de la capa. Los umbrales de activación en cada una de las capas fueron inicializados a cero. El número de épocas de entrenamiento se fijó a 650. Se inició con una tasa de aprendizaje de 0.001 la cual se disminuyó en un factor de 0.98 después de cada 2 épocas.

Para evaluar el rendimiento de la CNN, la red se entrenó desde cero, y la base de datos utilizada fue dividida en conjunto de entrenamiento y prueba. El conjunto de entrenamiento fue formado con el 70% de las imágenes de la base de datos y el 30% restante se utilizó para formar el conjunto de prueba; estos conjuntos fueron guardados en archivos tfrecord [TensorFlow, 2018]. La CNN fue entrenada y validada en una Workstation con 16 GB de memoria RAM, utilizando una GPU NVIDIA GeForce GTX TITAN X con 3072 núcleos y 12 GB de memoria, utilizando el marco de aprendizaje profundo TensorFlow [TensorFlow, 2018].

Implementación de la red en la Raspberry Pi 3

La Raspberry Pi 3 a través de una cámara web, captura la imagen a analizar en formato RGB, la imagen es redimensionada a un tamaño de 224x224x3 ajustándola al tamaño requerido por la capa de entrada de la CNN. La imagen redimensionada es enviada al NCS para determinar la categoría de los objetos que contiene. Finalmente, la Raspberry Pi 3, genera y reproduce una salida de audio que describe la clase a la que pertenece la imagen visualizada. El software eSpeak [eSpeak, 2018], fue instalado en el sistema operativo y utilizado para convertir el texto que describe la clase en audio, el cual es reproducido por la bocina. El software incluye diferentes voces y permite alterar sus características.

El NCS es el dispositivo en donde se carga la CNN preentrenada para la aplicación. El NCS recibe la imagen de la Raspberry Pi 3 y realiza la clasificación del objeto preponderante que contiene la imagen. La predicción es regresada a la Raspberry Pi 3 para que ésta genere y reproduzca la salida de audio que describe la clase del objeto visualizado.

En la figura 3, se muestra la integración de los elementos principales del prototipo propuesto, la Raspberry Pi 3 y el “Neural Compute Stick”. La programación de la

Raspberry Pi 3 se hizo en Python 2.7, usando la librería OpenCV 3.4. La Raspberry captura 18 fotogramas cada 3 segundos, tiempo en el cual, el NCS realiza 18 clasificaciones de las imágenes instantáneas. Si durante los 3 segundo, la salida del NCS es la misma clase, la Raspberry traduce en audio la etiqueta de la clase para describir el objeto encontrado en la imagen y guarda en disco una copia de la imagen junto con su etiqueta.

Figura 3 Prototipo funcional para clasificación de objetos en imágenes con salida de audio en un sistema embebido con red neuronal convolucional.

Procedimiento simple, basado en TensorFlow, para cargar la CNN preentrenada en el NCS; cabe aclarar que la CNN es diseñada y preentrenada en un servidor utilizando herramientas apropiadas para ese fin:

- En primer lugar, se entrena la CNN diseñada utilizando una base de datos para la aplicación deseada. El entrenamiento se realiza bajo el ambiente de trabajo, TensorFlow y Python. Al final del entrenamiento, TensorFlow genera y guarda tres archivos: .index, .meta y .data, los cuales contienen toda la información del entrenamiento.
- En segundo lugar, se realiza una transferencia de aprendizaje de la CNN entrenada, en donde se remueve de la red: el algoritmo de aprendizaje y las últimas dos capas “Dropout” y “Classification”. Con esto, la capa Softmax es declarada como la capa de salida de la red, generando como variable de salida *varOutput = sofmax()* la cual dará las predicciones de las distintas

clases. Al final de este paso, se generan y guardan nuevamente los tres archivos con las extensiones mencionadas en el punto anterior.

- Los tres archivos de inferencia generados en el punto 2, son utilizados para generar un archivo con extensión **.graph* a través del compilador del SDK (kit de desarrollo de software) del NCS. Esta tarea es realizada con el comando siguiente: *mvNCCompile MiRed_inference.meta -s 12 -in variable_entrada -on variable_salida -o MiRed_inference.graph*. Para más detalles sobre la compilación consultar [NC SDK, 2018].
- Una vez generado el archivo **graph* que contiene la CNN entrenada, se codifica para que sea cargado en el NCS; la Raspberry Pi 3 recibe las clasificaciones generadas por el NCS para cada imagen de consulta.

Los primeros tres puntos del procedimiento anterior se realizan en el servidor, el punto cuatro se implementa en la Raspberry Pi 3.

Base de datos

En este trabajo, se utilizó el conjunto de datos “Visual Object Classes Challenge 2012 (VOC2012)” [Everingham, 2018], la cual consta de 11,540 imágenes a color de alta resolución pertenecientes a 20 clases.

Cada clase contiene de 303 a 4,087 imágenes y la mayoría de las categorías tienen alrededor de 550 imágenes.

Se seleccionaron cinco clases (“carro”, “gato”, “perro”, “motocicleta” y “persona”), para formar un subconjunto con 8,140 imágenes; donde la clase con el menor número de imágenes es “motocicleta” con 526, y la clase con el mayor número de imágenes es “persona” con 4,087.

Para no lidiar con el desbalance de las clases del subconjunto de imágenes, se estandarizó el número de imágenes por clase, a la clase con el menor número de imágenes, para formar un conjunto de 2,630 imágenes; este es el conjunto base para el entrenamiento y prueba de la CNN. La estandarización se realizó de manera aleatoria.

Requisitos del prototipo

En la tabla 1 se resumen las características del Hardware y el Software utilizado para el desarrollo del prototipo propuesto.

Tabla 1 Características de Hardware y Software.

Hardware	Entrenamiento: • GPU NVIDIA GeForce GTX TITAN X, con 12 GB de memoria RAM y 3076 núcleos. • Workstation Dell T7600, Procesador Intel(R) Xeon(R), 16 Gb de Memoria RAM, 1 disco duro de 500 GB.
	Implementación: • Neural Compute Stick: USB 3, 4 GB LPDDR3, Precisión FP16. • Raspberry Pi 3 Model B: CPU Quad Core 1.2 GHz, Broadcom, BCM2837 de 64bits, 1 GB de RAM, Chip BCM43438 wireless LAN y Bluetooth, GPIO de 40 pines, 32 GB de memoria Micro SD. • Cámara Web. • Bocina con entrada plug 3.5 mm.
Software	Entrenamiento: • Sistema operativo Linux Ubuntu 16.04, kernel 4.12. • Python 2.7. • TensorFlow v1.4. • NVIDIA CUDA® 8.0. • NVIDIA cuDNN v5.1.
	Implementación: • Sistema operativo Raspbian Stretch. • Neural Compute SDK. • OpenCV 3.4. • eSpeak text to speech v18.

3. Resultados

Entrenamiento y prueba de la CNN

En la figura 4 se muestran las gráficas de exactitud de entrenamiento y prueba obtenidas para la red GoogleNet después de 650 épocas de entrenamiento. En entrenamiento, la CNN llega a una exactitud del 98.75% en las 650 épocas, mientras que en la fase de prueba la red logra un rendimiento máximo de 60.13%. Obsérvese que después de la época 450, la exactitud de entrenamiento se incrementa mientras que la exactitud de prueba permanece constante, lo que indica, que la red quedó

ajustada a características muy específicas de los datos de entrenamiento; es decir, la red sufre de sobreajuste.

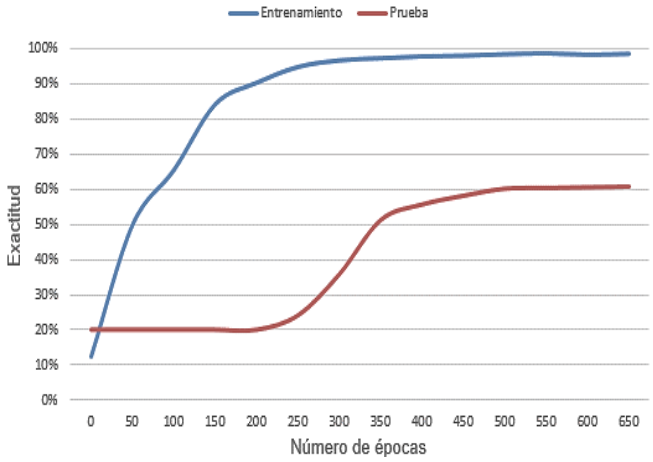

Figura 4 Gráficas de exactitud de la CNN entrenada en 650 épocas.

Para visualizar mejor el rendimiento de la CNN en la fase de prueba, en la figura 5, se muestra una matriz de confusión, en la cual cada fila representa el número de predicciones de cada clase, mientras que cada columna representa los ejemplos de una clase real.

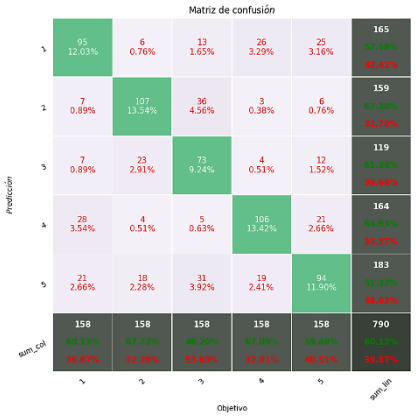

Figura 5 Matriz de confusión para la fase de prueba de la CNN.

Las celdas diagonales corresponden a los ejemplos que están clasificados correctamente. Las celdas fuera de la diagonal corresponden a ejemplos incorrectamente clasificados. En cada celda se muestran tanto el número de

ejemplos como el porcentaje del número total de ejemplos. La columna en el extremo derecho de la gráfica muestra los porcentajes de todos los ejemplos que se predice que pertenecen a cada clase que están clasificados correcta e incorrectamente. La fila en la parte inferior de la gráfica muestra los porcentajes de todos los ejemplos que pertenecen a cada clase que están clasificados correcta e incorrectamente. Finalmente, la celda en la parte inferior derecha de la gráfica muestra la precisión general. Las clases con el mejor rendimiento fueron “gato” (2) y “motocicleta” (4) con el 67.72% y 67.09% respectivamente; mientras que la clase con el peor rendimiento fue “perro” con el 46.20%. La red tiene problemas para distinguir entre “carro” y “motocicleta”, y entre “gato” y “perro”. La red también confunde “persona” con “carro” o “motocicleta”, esto debido a que una gran cantidad de imágenes de la clase “persona” contienen carros o motocicletas.

Implementación de la red en la Raspberry Pi 3

Las pruebas se llevaron a cabo en el estacionamiento de la universidad para detectar autos, personas y motocicletas; para detectar perros y gatos las pruebas se realizaron en diferentes hogares. Se experimentó con 10 pruebas por clase a una distancia de entre un metro y tres metros para clasificar a las personas. Para los carros y las motocicletas, la distancia fue de tres a cinco metros como máximo. En la figura 6 se muestran los resultados de las predicciones de las 5 clases, con imágenes reales, utilizando el prototipo.

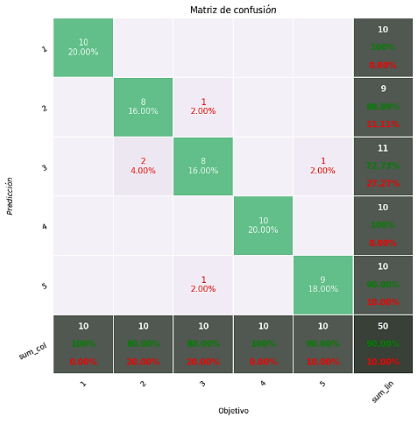

Figura 6 Matriz de confusión para la clasificación de imágenes utilizando el prototipo.

El prototipo logró predecir con una exactitud del 100% a las clases “carro” y “motocicleta”; mientras que para las clases “persona”, “gato” y “perro”, el éxito fue menor. La precisión de predicción total de la red en las pruebas realizadas fue del 90%; sin embargo, la red tiene problemas en la predicción de las clases “gato” y “perro” principalmente. La red tiene buen éxito de predicción cuando la imagen que capta la cámara contiene sólo una de las cinco clases, como se puede apreciar en las figuras 7 y 8, donde el prototipo la clasifica los objetos con una exactitud del 100%. Sin embargo, la red puede tener problemas de predicción con imágenes que contienen más de una de las cinco clases.

Figura 7 Imagen de Camioneta dentro del estacionamiento de la UAM-Azcapotzalco.

Figura 8 Imagen de un alumno dentro de un laboratorio de la UAM-Azcapotzalco.

4. Discusión

La implementación de redes neuronales convolucionales en sistemas embebidos de bajo costo, para aplicaciones de visión artificial, es relativamente fácil con el uso del NCS. La instalación del software para ambos dispositivos es relativamente sencilla, sin embargo, el entrenamiento y compilación (archivo *.graph) de la red son un poco complicados.

La cantidad insuficiente de imágenes utilizadas para el entrenamiento de una red tan profunda como GoogleNet, produce un rendimiento pobre en la fase de prueba. Entrenar la red con un número grande de épocas, no implica obtener un mejor rendimiento en prueba, pero sí puede provocar que la red se adapte cada vez más a los ejemplos de entrenamiento causando el sobreajuste de la red. La CNN alcanza su máximo rendimiento en prueba en aproximadamente en 450 épocas y permanece constante hasta el final de entrenamiento, señal del sobreajuste que está presentando. Para garantizar un entrenamiento eficiente y un aumento considerable en el rendimiento en la fase de prueba, es necesario utilizar un conjunto de entrenamiento lo suficientemente grande; que se podría obtener aplicando la técnica de aumento de datos, la cual puede aumentar el tamaño del conjunto de entrenamiento 10 veces o más, haciendo a la red más robusta para evitar el sobreajuste.

En los conjuntos de entrenamiento y prueba existen imágenes que pertenecen a una clase, pero incluyen elementos de otras clases, por ejemplo, en la figura 9 se muestra una imagen que pertenece a la clase “carro” que también incluye a las clases “persona”, “motocicleta” y “perro”. Imágenes como estas confunden mucho al clasificador y producen un rendimiento de prueba bajo.

Figura 9 Imagen que contiene cuatro de las cinco clases utilizadas.

Eliminar este tipo de imágenes del conjunto de datos, podría mejorar el rendimiento de la red; sin embargo, esto no sería una buena idea ya que este tipo de imágenes son muy comunes en calles y avenidas hoy en día. El clasificador desarrollado en

el prototipo, solo considera la clasificación de una clase a la vez, no una combinación de ellas. La reproducción en audio de la etiqueta del objeto clasificado, se realiza correctamente; sin embargo, se podría mejorar seleccionando diferentes tipos de voz y variando los parámetros de frecuencia y duración.

5. Conclusiones

Los resultados obtenidos en este trabajo, indican que tanto la profundidad de la CNN como el tamaño del conjunto de entrenamiento, son cruciales para que la red alcance un rendimiento alto en la fase de prueba; en otras palabras, para que la red tenga una buena capacidad de generalización.

El Neural Compute Stick, es un dispositivo muy confiable para la implementación de redes neuronales convolucionales profundas y su integración con sistemas embebidos de bajo costo, como la Raspberry Pi 3, es muy simple; sin embargo, hay que tener cuidado con la compilación y la descarga de la red.

El prototipo portátil desarrollado, tiene la capacidad de clasificar 5 clases de objetos diferentes y de reproducir una salida de audio describiendo la clase a la que pertenece el objeto. Sistemas con estas características, pueden ser de gran uso para desarrollar una gran cantidad de aplicaciones, por ejemplo, para ayudar a personas invidentes a encontrar objetos o para ayudar a niños de pre-escolar a aprender los nombres de los objetos, por citar algunos.

Como trabajo futuro se planea, evaluar el rendimiento del NCS con redes más profundas, por ejemplo: ResNet 152. Redes entrenadas con conjuntos de datos que contienen miles o millones de imágenes a color de alta resolución, desarrolladas para la detección de objetos.

6. Bibliografía y Referencias

[1] Caffe: http://caffe.berkeleyvision.org/, Agosto 2018.

[2] Ciresan, D, C., Meier, U., Masci, J., Gambardella, L., Schumidhuber, J., Flexible high performance convolutional neural networks for image classification. In Proceeding of the Twenty-Second International Joint Conference on Artificial Intelligence, Vol. 2, 1237-1242, 2011.

[3] eSpeak: http://espeak.sourceforge.net/commands.html, Abril 2018.

[4] Everingham, M. and Van~Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results:http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[5] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., Culurciello E., Hardware accelerated convolutional neural networks for synthetic vision systems. In Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), 257-260, 2010.

[6] Girshick, R., Donahue, J., Darrell, T., Malik, J., Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 580-587, 2014.

[7] He, K., Zhang, X., Ren, S., Sun, J., Deep residual learning for image recognition. In Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778, 2016.

[8] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications. CoRR, vol. abs/1704.04861, 2017.

[9] Karpathy, A., Fei-Fei, L., Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3128-3137, 2015.

[10] Krizhevsky, A., Sutskever, I., Hilton, G., ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, 1097-1105, 2012.

[11] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., Gradient-based learning applied to document recognition. In Proceedings of the IEEE, Vol. 86, Issue 11, 2278-2324, 1998.

[12] Mollahosseini, A., Chan, D., Mahoor, M, H., Going deeper in facial expression recognition using deep neural networks. In Proceeding IEEE Conference on Applications of Computer Vision (WACV), 1-10, 2016.

[13] Neural Compute Application Zoo (NC App Zoo): https://github.com /movidius/ncappzoo/.

[14] NC SDK, Intel® Movidius™: https://movidius.github.io/ncsdk/tf_compile_ guidance.html.

[15] NCS Quick Star, Intel® Movidius™: https://developer.movidius.com/start.

[16] Peemen, M., Setio, A., Mesman, B., Corporaal, H., Memory-centric accelerator design for convolutional neural networks. 31st International Conference on Computer Design, 13-19, 2013.

[17] Prasoon, A., Petersen, K., Igel, C., Lauze, F., Dam, E., Nielsen, M., Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network. In International Conference on Medical Image Computing and Computer-Assisted Intervention, 246-253, 2013.

[18] Raspberry Pi: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[19] Sankaradas, M., Jakkula, V., Cadambi, S., Chakradhar, S., Durdanovic, I., Cosatto, E., Graf, H., A massively parallel coprocessor for convolutional neural networks. 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 53-60, 2009.

[20] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y., OverFeat: Integrated recognition, localization and detection using convolutional networks. Journal: arXiv preprint arXiv: 1312.6229, 2014: https://arxiv.org/pdf/1312.6229.pdf.

[21] Stanford Vision Lab, Stanford University, Imagenet. http://imagenet.org, Octubre 2017.

[22] Strigl, D., Kofler, K., Podlipnig, S., Performance and scalability of gpu-based convolutional neural networks. In Proceeding 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 317-324, 2010.

[23] Suriyal S., Druzgalski C., Gautam K. Mobile assisted diabetic retinopathy detection using deep neural network. 2018 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE), marzo 2018.

[24] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. Going Deeper with Convolutions. In Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June, 2015.

[25] Tafjira, N, B., Shun-Feng, S., Towards self-driving car using convolutional neural network and road lane detector. 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro--Mechanical System, and Information Technology (ICACOMIT), 65-69, 2017.

[26] TensorFlow: https://www.tensorflow.or/.

[27] Vasilache, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S., LeCun, Y., Fast convolutional net with fbfft: A gpu performance evaluation. arXiv preprint arXiv:1412.7580, 2014: https://arxiv.org/pdf/1412.7580.pdf.

[28] Xu, X., Amaro, J., Caulfield, S., Forembski, A., Falcao, G., and Maloney, D. Convolutional Neural Network on Neural Compute Stick for Voxelized Point-clouds Classification. 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2017.