Reconocimiento de Objetos con CNN en entornos de navegación autónoma

1st Nicolás Urbano Pintos
División Radar Láser
CITEDEF
Villa Martelli, Argentina
urbano.nicolas@gmail.com

***Abstract*—En este trabajo se propone reconocer objetos a través de redes neuronales de convolución, utilizando el modelo VGG16, y entrenando con el dataset CIFAR10. Se logró una precisión cercana al 90% con el conjunto de evaluación de CIFAR10. Se muestran los resultados en la detección de objetos de imágenes obtenidas del dataset KITTI de navegación autónoma.**

***Index Terms*—CNN, CIFAR10, PYTORCH**

I. Introducción

Las redes neuronales de convolución (CNN – Convolutional Neural Network) son algoritmos de aprendizaje profundo [1], los cuales tienen como entrada una imagen, y se les asignan pesos y bias a los aspectos y/o objetos de la imagen, de modo de poder diferenciarlos entre si. Las CNN fueron inspiradas en la organización de la corteza visual del cerebro humano. Neuronas individuales responden a estímulos en una región restringida del campo visual. Una colección de dichos campos son superpuestos para cubrir el área visual de forma completa. La convolución 2D consta de multiplicar regiones de píxeles de la imagen por una matriz denominada kernel, la cual tiene un tamaño determinado, por ejemplo 3x3 o 5x5, y está compuesta por pesos. Luego de multiplicar cada píxel de la región por el peso correspondiente, se suman todos los resultados obteniendo como salida un nuevo píxel. El kernel se desplaza por toda la imagen. En la 1 se observa como la

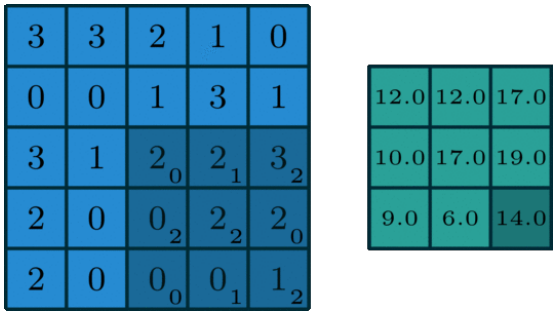

Fig. 1. Convolución de una matriz de 5x5 con un kernel de 3x3

matriz original, se convierte en una nueva matriz de 3x3. Cada píxel de la nueva imagen representa a una región de la imagen original. De acuerdo al tipo de kernel utilizado, se obtienen diferentes características de bajo nivel, como lineas y bordes. Si se utilizan a continuación otras capas de convolución se obtienen características de alto nivel, como curvas y texturas. Y si se utilizan aún mas capas se obtienen características de muy alto nivel como partes o patrones. Existen 2 técnicas utilizadas en las capas de convolución: Padding y Strides.

A. Padding

Al aplicar el kernel sobre las regiones de píxeles que están en los bordes, los píxeles del borde nunca quedan centrados en el píxel. El padding resuelve esto utilizando píxeles extra en los bordes, denominados píxeles falsos, que generalmente son 0. De este modo la información de los píxeles del borde puede quedar en el centro del kernel.

B. Striding

Al aplicar una capa de convolución se desea que la matriz de salida tenga un tamaño menor a la de la entrada. En la CNN es común que la salida se reduzca en dimensión, pero que aumente en profundidad. Esto se logra utilizando una capa de Pooling, generalmente un kernel de 2x2 que toma el promedio o el máximo de la región de píxeles. Otra manera de utilizar esta técnica es saltando ubicaciones de píxeles. En el caso de la convolución de múltiples canales, por ejemplo una imagen de 3 canales (RGB), se utiliza una colección de kernel's, en este caso 3. Se convoluciona cada kernel por separado, y luego se suman cada uno de los píxeles obtenidos entre si. De modo de obtener un solo canal. Además, se utiliza un término de bias, que es sumado para obtener la salida.

Fig. 2. Arquitectura de modelo VGG16

II. Configuración de la CNN

A. Arquitectura

Las redes de aprendizaje profundo tienen múltiples capas, las cuales puede aprender características en varios niveles de abstracción. Por ejemplo, si se entrena a una CNN para clasificar imágenes, la primer capa será capaz de reconocer aspectos básicos como líneas, la próxima capa reconocerá

ejes y/o bordes, y la próxima una colección de formas, como curvas y texturas. Así sucesivamente hasta poder reconocer características de alto orden como objetos (camiones, barcos, perros, etc.). En este trabajo se utiliza como modelo a la variación de la red VGG propuesta por Lui et. Al [2].

B. Regularización

Las redes neuronales profundas con gran cantidad de parámetros son sistemas de aprendizaje de máquina muy poderosos. Sin embargo, el sobre ajuste 'overfitting´ es un problema recurrente en muchas redes. Por ese motivo existen diversas técnicas para evitarlo, como Dropout, Kernel regularizer y BatchNormalization [3]. En este caso, se utiliza BatchNormalization, quién normaliza la activación de la capa anterior de cada lote, por ejemplo aplicando una transformación que mantiene el promedio de activación cercano a 0 y la desviación estándar cercana a 1. Esto direccional el problema del desplazamiento de la co-varianza interna. También actúa como regularizador, en algunos casos eliminando la necesidad de utilizar Dropout. Logra la misma precisión con menores pasos de entrenamiento por lo tanto acelera el proceso de entrenamiento.

C. Optimizador

Se encarga de generar pesos que ajusten de mejor manera el modelo. Calcula el gradiente de la función de pérdidas por cada peso, a través de la derivada parcial. El objetivo es minimizar el error, por lo tanto los pesos se modifican en dirección negativa del gradiente.

$$W_t + 1 = W_t - \frac{df(coste)}{dW} * lr \tag{1}$$

El descenso de gradiente estocástico mantiene una tasa de aprendizaje única para actualizar todos los pesos, y la tasa no cambia durante el proceso de entrenamiento. Se utiliza el algoritmo de Adam (Adaptive moment estimation) [4], el cual es una combinación de AdaGrad (Algoritmo de gradiente adaptativo) y RMSprop (Propagación cuadrática media). Se tiene un factor de entrenamiento por parámetro. Cada factor de entrenamiento se ve afectado por la media del momentum del gradiente.

III. Entrenamiento

A. Dataset

El dataset CIFAR-10 [5] consiste de 60 mil imágenes color de 32x32 píxeles, con 6000 mil imágenes por cada clase. Hay 50 mil imágenes para entrenamiento y 10 mil para evaluación. Las clases se encuentran etiquetadas de acuerdo a la figura

Se entrenó el modelo sobre el dataset CIFAR-10 con 3 learning rate diferentes 0.1, 0.01 y 0.001. De las 50 mil imágenes del conjunto de entrenamiento, se separaron 5 mil de forma aleatoria para utilizarlas como validación. Se utilizó un tamaño de batch de 128. A las imágenes del dataset se le aplicó una normalización de histograma, la transformación a tensor de pytorch, y una función de volteo aleatorio y recorte aleatorio para aumentar los datos. Se corrió el programa

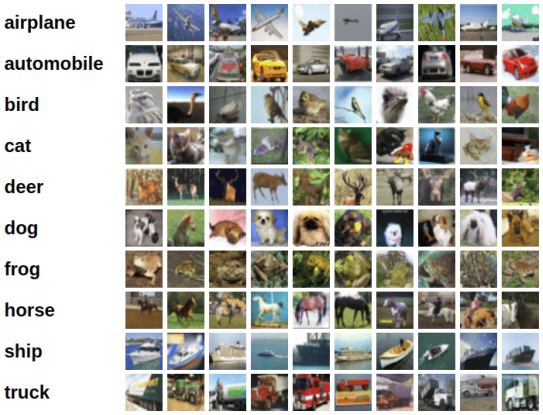

Fig. 3. Clases de dataset CIFAR-10

codificado en python con la librería pytorch en un entorno de Google Colaboratory con GPU compatible con CUDA.

IV. Resultados

Luego de entrenar con diferentes tasa de aprendizaje (lr) se observa la evolución de la precisión de validación para cada época.

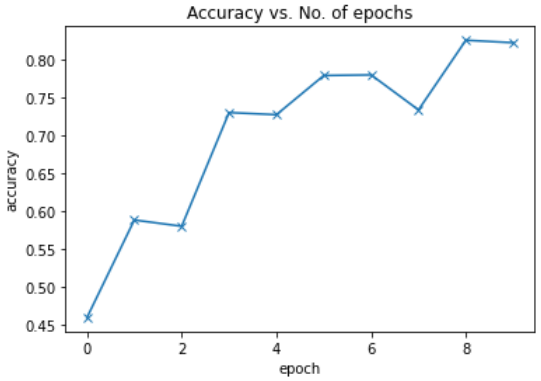

Fig. 4. Precisión de validación por epoca con lr=0.001

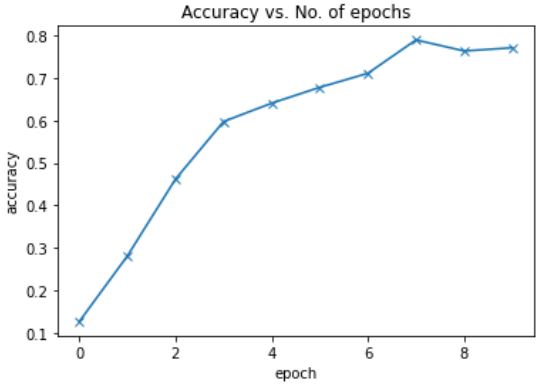

Fig. 5. Precisión de validación por época con lr=0.01

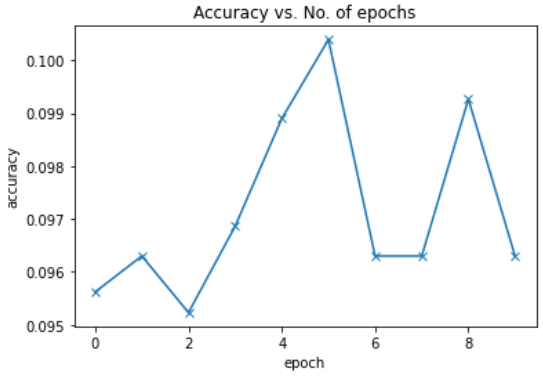

Fig. 6. Precisión de validación por época con lr=0.1

TABLE I
PRECISIÓN DE VALIDACIÓN PARA DIFERENTES LR

Época\lr	**0.001**	**0.01**	**0.1**
	val_acc	***val_acc***	***val_acc***
[0]	0.4714	0.1270	0.0956
[1]	0.6264	0.2814	0.0963
[2]	0.6298	0.4621	0.0952
[3]	0.7398	0.5975	0.0969
[4]	0.7205	0.6407	0.0989
[5]	0.7692	0.6780	0.1004
[6]	0.7946	0.7110	0.0963
[7]	0.7803	0.7895	0.0963
[8]	0.8094	0.7635	0.0993
[9]	0.8148	0.7710	0.0963

En la tabla I se observa que con lr igual 0.001 se obtiene la máxima precisión de validación en 10 épocas. Por este motivo se seleccionó este lr para entrenar con 50 épocas. Donde se obtuvo una precisión sobre el datastet de validación de casi 90%. En la 8 se observa que no hubo uno "overfitting",

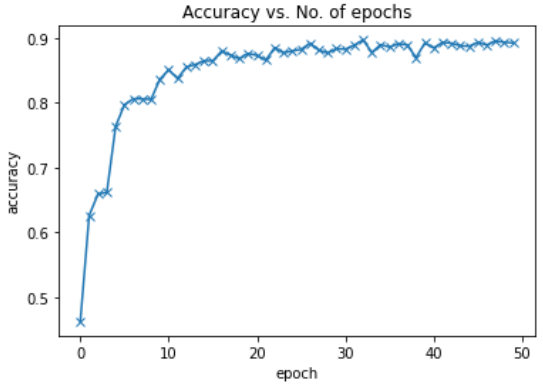

Fig. 7. Precisión de validación por época con lr=0.001 en 50 épocas

ya que las perdidas de entrenamiento y de validación siguen disminuyendo en cada época. A continuación se observan resultados de predicción de clase con el conjunto de evaluación del CIFAR 10. A continuación se observan resultados de

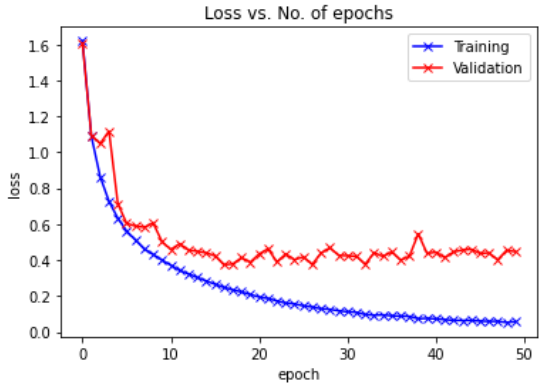

Fig. 8. Pérdidas de validación y entrenamiento

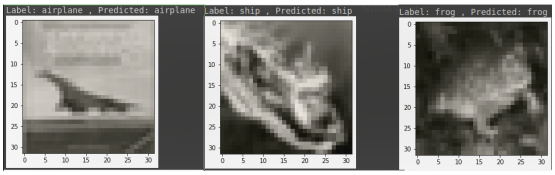

Fig. 9. Predicción de clase del conjunto de evaluación de CIFAR-10

predicción de clase con imágenes obtenidas de internet. Por

Fig. 10. Predicción de clase de imágenes obtenidas en internet

último se observan resultados de predicción de clase con imágenes obtenidas del dataset KITTI [6].

Fig. 11. Predicción de clase de imágenes obtenidas del dataset KITTI

V. CONCLUSIONES

Se logró una precisión de casi el 90% sobre con el conjunto de validación del dataset CIFAR-10. Se observa que el modelo predice satisfactoriamente sobre el conjunto de evaluación provisto por el dataset. Respecto a las imágenes obtenidas de internet, aquellas que no presentan fondos muy densos, se predice correctamente la clase. En las imágenes obtenidas del datastet KITTI, el modelo no logra predecir con precisión los objetos que aparecen en la imagen, esto se debe principalmente a que las escenas tienen varios objetos, y fondos densos. Respecto al tiempo de ejecución del modelo, que es del orden de los milisegundos, lo hace muy lento para realizar un sistema

de decisión para un vehículos autónomos, ya que este tiene que tomar decisiones en tiempo real.

VI. Trabajo a futuro

Para mejorar los resultados en los entornos de navegación autónoma, los cuales presentan un escenario con varios objetos, y con fondos diversos, es necesario recortar la imagen. Para solucionarlo es necesario realizar una etapa previa de localización de objeto, a través de cuadros delimitadores (bounding boxes).

Además, para reducir el tiempo de ejecución de la predicción del modelo, se plantea a futuro adaptar el modelo VGG16 a un modelo con pesos binarios, de modo tal de implementarlo en una FPGA [7], y de este modo disminuir los tiempos de ejecución.

References

[1] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks" on 2017 International Conference on Communication and Signal Processing (ICCSP), 2017, pp. 0588-0592, doi: 10.1109/ICCSP.2017.8286426.

[2] Liu, Shuying and Deng, Weihong on 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), "Very deep convolutional neural network based image classification using small training sample size", 2015, 10.1109/ACPR.2015.7486599.

[3] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift" on International conference on machine learning. PMLR, 2015.

[4] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization" arXiv preprint arXiv:1412.6980, 2014.

[5] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." 2009.

[6] Andreas Geiger and Philip Lenz and Raquel Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite" on Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[7] Umuroglu, Yaman, et al. "Finn: A framework for fast, scalable binarized neural network inference." Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays. 2017.