Universidad de O'Higgins

Tarea N°3:
Redes Neuronales Convolucionales

Cristopher Acevedo Guajardo

COM4402 – Introducción a Inteligencia Artificial

Escuela de Ingeniería, Universidad de O'Higgins

11, Noviembre, 2023

***Resumen*— En este informe sobre la tarea número 3 del curso se presenta el entrenamiento y la aplicación de una red neuronal convolucional utilizando TensorFlow y utilizando el conjunto de datos MNIST (Modified National Institute of Standards and Technology database) que consta de imágenes en escala de grises para analizar y comparar el rendimiento de la red en un problema de clasificación de dígitos al utilizar Max-Pooling en comparación a no utilizar la operación de pooling para reducir la cantidad de datos entre capas de la red. Se utilizan las métricas de *validation-loss* y *accuracy* para evaluar el rendimiento de las redes neuronales convolucionales y también se evalúa el rendimiento de la red con Max-Pooling al utilizar el conjunto de datos CIFAR-10 que consta de imágenes a color compuesta por 10 clases. Las pruebas realizadas indican que, para el problema de clasificar dígitos manuscritos, la red neuronal convolucional implementada tiene un rendimiento casi igual que al utilizar la misma red convolucional, pero sin Max-Pooling, y para el conjunto de datos CIFAR-10, la red no pudo entrenarse/ajustarse al conjunto de datos con imágenes en una escala de colores RGB.**

I. Introducción

Las redes neuronales convolucionales son un tipo de red neuronal capaces de aprender y reconocer características o "features" relevantes en una imagen para luego resolver un problema de clasificación.

El objetivo principal de esta tarea es evaluar el rendimiento de una red neuronal convolucional en un conjunto de datos en escala de grises y con Max-Pooling y compararla con el rendimiento de una red convolucional sin la opción de Max-Pooling, y luego usar la misma red (con Max-Pooling) para entrenar y evaluar un conjunto de datos en una distinta escala de colores.

Se utilizará la base de datos MNIST (Modified National Institute of Standards and Technology database) otorgada por el equipo docente del curso, que contiene 10 clases (dígitos manuscritos del 0 al 9) con imágenes de 28x28 píxeles, con un conjunto de entrenamiento de 60.000 muestras y un conjunto de pruebas de 10.000 muestras.

También, se utilizará la base de datos CIFAR-10, otorgada por el equipo docente del curso, que contiene 10 clases (tipos de prendas de vestir) con la misma cantidad de muestras para el conjunto de datos de entrenamiento y de prueba, pero con la diferencia de que las imágenes en este conjunto de datos son a color, es decir, en una escala de colores RGB.

Se evaluará el rendimiento de cada modelo de red neuronal convolucional utilizando las de *validation-loss* y accuracy.

II. Marco teórico

Las redes neuronales convolucionales (CNN o ConvNet) son una clase de red neuronal profunda (utilizadas en Deep Learning) utilizadas para el procesamiento de datos bidimensionales, y más frecuentemente para el análisis de imágenes.

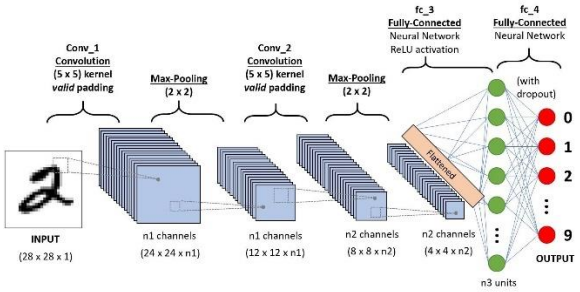
La red utiliza capas de "convolución", que aplican filtros en lugares específicos de la imagen y así identificar y aprender patrones y características dentro de una imagen, y luego implementa capas de "pooling" que son capas que permiten reducir la dimensionalidad de las características extraídas de la imagen (en la capa de convolución) y así hacer la red más eficiente y con un menor costo computacional.

Dentro de la estructura o arquitectura de una red neuronal convolucional, tendremos como input una imagen en donde cada pixel de la imagen será una característica (por ejemplo, una imagen de 10x10 píxeles tendrá 100 inputs), donde se aplicará la convolución, es decir, se aplicarán filtros a esa imagen para crear mapas de características o "feature maps", y luego se aplicará el Max-Pooling (también existen otras opciones de pooling, como Average-Pooling) en donde se extren las características más relevantes o con más información de la capa de

convolución y se reduce la dimensionalidad del mapa de características, y luego se conectan a una capa neuronal conectada (o “fully-connected layer) que realiza las clasificaciones.

Al igual que en las redes neuronales (no convolucionales), usamos una función de activación, principalmente ReLU después de cada capa de convolución y de pooling para introducir no-linealidad en la red, una función de optimización y una función de error, en donde, dependiendo de la salida identificada por la red comparada con la salida real, se ajustan en cada época los pesos de los filtros en las capas de convolución.

ILUSTRACIÓN 1
AQUITECTURA DE UNA RED NEURONAL CONVOLUCIONAL



FUENTE: “WHATS IS THE CONVOLUTIONAL NEURAL NETWORK ARCHITECTUURE?”, ANALYTICS VIDHYA, HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2020/10/WHAT-IS-THE-CONVOLUTIONAL-NEURAL-NETWORK-ARCHITECTURE/

III. METODOLOGÍA

Esta sección describe en detalle el proceso y la metodología que se aplicó para evaluar el rendimiento de los modelos de redes neuronales convolucionales en la base de datos MNIST y CIFAR-10.

A. Carga del conjunto de datos

La primera base de datos utilizada en esta tarea es el MNIST, que está incluida en la librería Keras, y cuyo código fue otorgado por el equipo docente en el código base.

TABLA I
CARGA DE LA BASE DE DATOS AL PROGRAMA

```
# Mnist wrapper
from keras.datasets import mnist


# Código para cargar la base de datos MNIST
(x_train, y_train), (x_test, y_test) =
mnist.load_data()
```

B. Preprocesamiento de los datos

Se implementó una función para normalizar los datos del conjunto de datos. El conjunto de datos del MNIST contiene imágenes de dígitos manuscritos en escala de grises, por lo que la escala de los colores de cada píxel se encuentra en escala de 0 a 255. En la función se divide cada dato (píxel, en este caso) en 255, así los valores estarán en una escala de 0 a 1.

TABLA II
FUNCIÓN PARA NORMALIZAR LOS DATOS

```
def normalize_images(images):
    """Normalizar las imágenes de entrada.
    """
    # Normalizar la imagen aquí
    images = images / 255.0
    return images
```

C. Ampliación de la dimensión de entrada

Al cargar el conjunto de datos del MNIST, cada dígito manuscrito esta representado por una matriz de los pixeles correspondientes de cada imagen, es decir, de tamaño (28,28), y la red convolucional utiliza el concepto de canales de color y mapas de características, en donde indica el color (en la escala normalizada de 0 a 1) del pixel, por lo que se tuvo que transformar esta entrada de (28,28) a (28,28,1) para el correcto funcionamiento de la red.

TABLA III
AMPLIACIÓN DE LA DIMENSIÓN DE ENTRADA

```
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
```

D. Procesamiento de los labels

Para clasificar los dígitos, se utilizó la técnica de *one-hot encoding* para representar las salidas de los objetivos, tanto en el conjunto de datos MNIST como CIFA-10. Esta técnica es una solución robusta, sencilla y eficiente para representar los objetivos o salidas multicategóricas.

Como el conjunto de datos se compone de salidas *y* con números entre 0 a 9 con 10 clases, entonces con esta técnica se crea un nuevo vector *y'* con el mismo número de filas que el vector de salidas original (es decir, el total de ejemplos) y con un número de columnas igual al total de clases (10, en este caso).

Luego, respecto al valor de la salida que se tenga en cada fila en el vector de salidas (o labels) original *y*, se fija el valor de 1 en la misma fila en el vector de salidas codificado *y'* en la posición que corresponda al valor de la salida en el vector *y* (con indexación 0), y se dejan en 0 en las demás columnas.

Por ejemplo, si en la primera fila del vector de salida original *y* la clase es 5, entonces en el vector de salidas codificado *y'* se fija el valor de 1 en la columna correspondiente al 5, es decir, en la posición 6 (con indexación 0), y se dejan en 0 las demás columnas.

TABLA IV
ONE-HOT ENCODING

```
def one hot(vector, number classes):
    one_hot = []
    for i in vector:
      one_hot_vector = np.zeros(number_classes)
      one_hot_vector[i] = 1
      one_hot.append(one_hot_vector)
    return np.array(one_hot)
```

E. Construcción de la red neuronal convolucional y Max-Pooling

Se implementaron dos funciones para dos redes neuronales convolucionales distintas.

La primera función implementa una red neuronal convolucional con dos capas de convolución, la primera con 32 filtros, un kernel-size de (3,3), un número de strides de 1, cero padding y con función de activación ReLU. La segunda capa de convolución con las mismas características, pero con 64 filtros. Después de las dos capas de convolución, se aplica una capa de max-pooling, con un pool-size de (2,2), strides con la opción "None", y cero padding. Luego, se "aplanaron" los datos con la función *Flatten()*, para así utilizar los datos como una columna de inputs para la capa neuronal conectada (*fully-connected layer*) que hizo las clasificaciones, con 128 unidades y función de activación ReLU. Se utilizó la función *softmax* para la salida, y con la función de pérdida de *Cross-Entropy*, y con un batch-size de 128 y 30 épocas para entrenar la red.

TABLA V
RED NEURONAL CONVOLUCIONAL CON MAX-POOLING

```
def net_1(sample_shape, nb_classes):
    input_x = Input(shape=sample_shape)
    x = tf.keras.layers.Conv2D(filters=32,
kernel_size=(3,3), strides=1, padding="valid",
activation='relu')(input_x)
    x = tf.keras.layers.Conv2D(filters=64,
kernel_size=3, strides=1, padding="valid",
activation='relu')(x)
    x = MaxPooling2D(pool_size=(2,2),
strides=None, padding="valid")(x)
    x = Flatten()(x)
    x = Dense(units=128, activation='relu')(x)
    probabilities = Dense(nb_classes,
activation='softmax')(x)
    model = Model(inputs=input_x,
outputs=probabilities)
    return model
```

La segunda red neuronal se implementó con la misma estructura (mismas capas convolucionales y funciones) pero sin la capa de Max-Pooling, y con un batch-size de 128 y 30 épocas para entrenar la red.

TABLA VI
RED NEURONAL CONVOLUCIONAL SIN MAX-POOLING

```
def net_2(sample_shape, nb_classes):
    input_x = Input(shape=sample_shape)
    x = tf.keras.layers.Conv2D(filters=32,
kernel_size=(3,3), strides=1, padding="valid",
activation='relu')(input_x)
    x = tf.keras.layers.Conv2D(filters=64,
kernel_size=3, strides=2, padding="valid",
activation='relu')(x)
    x = Flatten()(x)
    x = Dense(units=128, activation='relu')(x)
    probabilities = Dense(nb_classes,
activation='softmax')(x)
```

```
    model = Model(inputs=input_x,
outputs=probabilities)
    return model
```

F. Evaluación de las redes

Se evaluaron las redes utilizando las métricas de *validation-loss* y *accuracy* para cada época, y se implementó un gráfico para ilustrar visualmente la disminución del *validation-loss* y el aumento del -*accuracy* durante cada época, y luego del entrenamiento se utilizaron las mismas métricas para evaluar el rendimiento de la red en el conjunto de pruebas de los conjunto de datos, utilizando la siguiente implementación:

TABLA VIII
EVALUACIÓN DE LA RED EN EL CONJUNTO DE PRUEBAS

```
score = model.evaluate(x_test, y_test,
verbose=0)
```

El conjunto de datos del MNIST se evaluó para ambas estructuras de redes neuronales y se compararon los resultados obtenidos y la diferencia entre usar max-pooling y no usarlo.

Para el conjunto de datos CIFAR-10, se utilizó la red neuronal convolucional con la capa de max-pooling, y se evaluó el rendimiento de la red en este conjunto de datos con las mismas métricas, pero evaluamos el rendimiento para distintos optimizadores, utilizando “Adadelta”, “Adagrad” y “Adam”.

IV. RESULTADOS

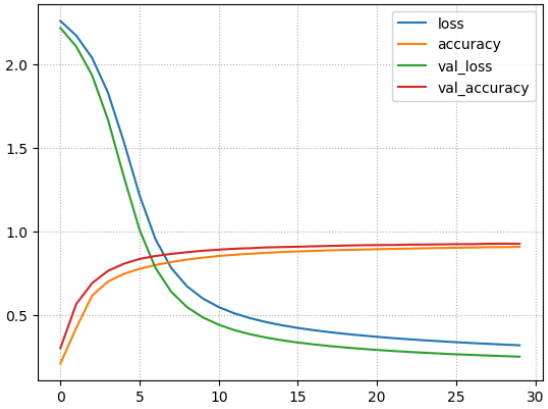
Los resultados obtenidos luego de haber realizado el entrenamiento y la evaluación de las dos estructuras de redes neuronales convolucionales, es decir, con la capa max-pooling y sin la capa max-pooling, con el conjunto de datos MNIST, fueron los siguientes:

A. *Red Neuronal Convolucional con Max-Pooling en el conjunto de datos MNIST*

TABLA IX
RENDIMIENTO DE LA RED EN EL CONJUNTO DE PRUEBA

# of epochs	Test Loss	Accuracy
30	0.2894	0.9200

LOSS Y ACCURACY EN EL CONJUNTO DE VALIDACIÓN

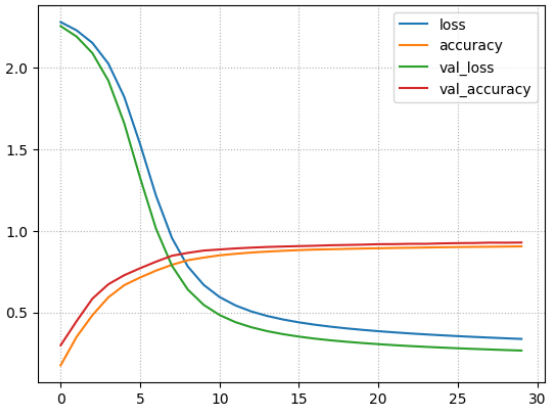


B. *Red Neuronal Convolucional sin Max-Pooling en el conjunto de datos MNIST*

TABLA X
RENDIMIENTO DE LA RED EN EL CONJUNTO DE PRUEBA

# of epochs	Test Loss	Accuracy
30	0.3075	0.9132

LOSS Y ACCURACY EN EL CONJUNTO DE VALIDACIÓN

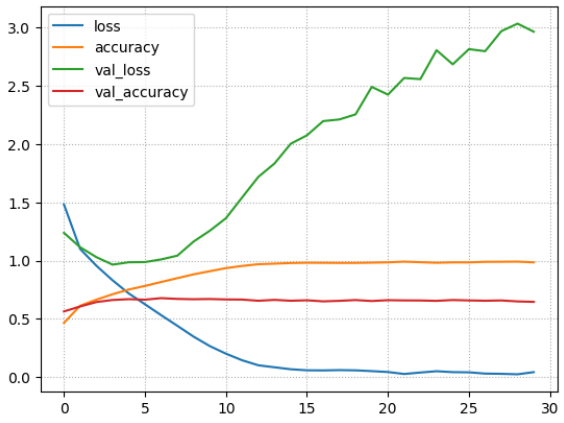


C. *Red Neuronal Convolucional con Max-Pooling en el conjunto de datos CIFAR-10*

V. TABLA IX
RENDIMIENTO DE LA RED EN EL CONJUNTO DE PRUEBA

# of epochs	Test Loss	Accuracy
30	2.9817	0.6519

Loss y Accuracy en el conjunto de validación



VI. Análisis

A. *Análisis de los distintos modelos utilizados*

Al utilizar el conjunto de datos MNIST, se pudo observar un rendimiento similar entre los dos modelos de redes neuronales convolucionales utilizados, es decir, con la capa de max-pooling y sin la capa de max-pooling.

En el caso de la red con max-pooling, con un batch size de 128 y 30 épocas, se obtuvo en el conjunto de pruebas un *test-loss* de 0.2894 y un *accuracy* de un 92%.

Para el caso de la red sin la capa de max-pooling, con un batch size de 128 y 30 épocas, se obtuvo en el conjunto de pruebas un *test-loss* de 0.3075 y un *accuracy* de un 91.32%.

En ambos casos, es decir, para ambas estructuras o modelos (con max-pooling y sin max-pooling), se pudo observar un buen rendimiento en general en el conjunto de prueba, pero aún así existe una pequeña diferencia en el *validation loss* y en el *accuracy* entre ambos, más precisamente, una diferencia de 0.0169 en el *valiadtion loss* y una dferencia mínima entre los *accuracy* de ambos modelos, pero el modelo con max-pooling siendo de un 92% y el modelo sin max-pooling estando más cerca del 91%, además, se observó que la convergencia o disminución del *validation loss* y el aumento del *accuracy* es levemente más rápida en el modelo con la capa de max-pooling, lo que lo lleva a ser un modelo más rápido y, por lo tanto, menos costoso computacionalmente, debido a la reducción de los feature maps que realiza el max-pooling, y para este caso, el modelo con max-pooling otorgó un rendimiento levemente mejor.

En el caso del conjunto de datos CIFAR-10, utilizando el modelo con max-pooling, se pudo observar un bajo o pobre rendimiento de la red al clasificar los datos.

Se pudo observar que el *validation loss* disminuye en las primeras épocas, pero rápidamente comienza a aumentar en las siguientes épocas (desde la 5 aproximadamente), divergiendo durante cada época que pasa. Así mismo, el *accuracy* alcanza su valor máximo alrededor del 65% en el conjunto de validación, y luego fluctúa entre los el 64% y el 66% en las épocas siguientes.

En el conjunto de pruebas se observó un test-loss de 2.9817, y un *accuracy* de un 65.19%.

Estos resultados indican un mal o pobre funcionamiento del modelo de utilizado (con max-pooling) para clasificar las imágenes del conjunto de datos CIFAR-10. Recordemos que este conjunto de datos está en una escala de colores, es decir, en una escala RGB, donde se debe analizar cada pixel respecto a los valores que tengan en éste cada color, tomando tres valores distintos: para rojo, para verde y para azul, utilizando una matriz del tipo (32, 32, 3), con 3 capas de color, a diferencia de la que habíamos utilizado al principio, que solo tenía una capa correspondiente a la escala de grises, por lo que el modelo o la función que definimos para el modelo/estructura de la red neuronal convolucional no era el adecuado para el este conjunto de datos.

B. *Análisis de los optimizadores*

Durante los distintos entrenamientos en el conjunto de datos del CIFAR-10 se utilizaron los optimizadores “Adadelta”, “Adagrad” y “Adam”.

Entre estos, a pesar de que ocurrió el mismo fenómeno de divergencia del *validation loss* y del bajo *test loss* y *accuracy* en el conjunto de prueba, el optimizador que entregó un mejor rendimiento en las métricas utilizadas fue el optimizador "Adam", cuyos resultados fueron reportados en la sección IV. El mismo optimizador ("Adam") fue el que otorgó un mejor rendimiento al evaluar el rendimiento de los modelos en el conjunto de datos MNIST.

C. *Análisis de overfitting*

En el conjunto de datos MNIST, al utilizar ambos modelos, es decir, las redes con max-pooling y sin max-pooling, se observó que el *test loss* y el *accuracy* en el conjunto de pruebas es mayor que los obtenidos en el conjunto de entrenamiento, por lo tanto, no hay evidencia de overfitting en los modelos para el conjunto de datos MNIST utilizando 30 épocas.

Por otro lado, en el conjunto de datos CIFAR-10 el *train loss* y el *accuracy* en el conjunto de entrenamiento fueron mucho mayor que en el conjunto de pruebas, teniendo un *train loss* de 0.0342 y un *accuracy* de 0.9888 en el conjunto de entrenamiento, mientras que en el conjunto de pruebas fue de 2.9817 y 65.18%, respectivamente, donde se evidenció un gran overfitting.

VII. CONCLUSIONES GENERALES

La estructura o modelos de redes neuronales convolucionales implementados tuvieron un buen rendimiento en general en el conjunto de datos de MNIST, siendo aproximadamente de 92% entre ambos modelos.

Aunque la diferencia entre ambos modelos es mínima, esto puede tener un gran impacto en otras aplicaciones. Por ejemplo, si se tiene el problema de identificar la existencia de algún tumor (o enfermedad) en base a imágenes como radiografías o similares, o si se está trabajando en la detección de objetos para un vehículo autónomo, esa pequeña diferencia en el *accuracy* puede ser una gran diferencia al momento de identificar o clasificar una enfermedad o en la detección de personas o árboles en el caso de un vehículo autónomo, por lo que el modelo con Max-Pooling podría ser una mejor opción.

Si bien los rendimientos están cerca del 92%, para llevar un modelo a la realidad puede ser necesario tener un mayor porcentaje o valor en el *accuracy* de el modelo, es decir, mejorar el rendimiento del modelo. Para esto, puede ser esencial para la red tener un conjunto de datos más grandes y poder tener muchas mas imágenes para aprender las características de imágenes más variadas y en distintos "contextos", y así, poder generar mejores features maps, y encontrar más detalles en las características de las imágenes que haga posible que el modelo tenga un mejor "entendimiento" de la forma de los objetos, y así poder identificar de mejor manera los objetos en distintas situaciones, formas, colores, entre otros, y aumentando así aún más el *accuracy* en los conjuntos de prueba.

También se pueden implementar más capas profundas (capas de convolución) para garantizar features maps más precisos, o cambiando el batch size (dependiendo del conjunto de datos con el que se esté trabajando) y el número de épocas con el cual se entrena la red, tendiendo siempre en consideración el no generar overfitting en el modelo.

En el conjunto de datos CIFAR-10 es necesario implementar otro modelo o estructura de red neuronal convolucional, ya que, al estar en escala RGB (color), las imágenes tienen tres escalas de colores: escala de rojos, escala de verdes y escala de azul, y la red que se implementó estaba diseñada para una sola escala de colores: grises.

BIBLIOGRAFÍA Y REFERENCIA

[1] Ignacio Bugeño, Alfonso Ehijo, Felipe Gómez, Camila Rapu, Código base Tarea 3, Escuela de Ingeniería, Universidad de O'Higgins. https://colab.research.google.com/drive/1K2JVSEu1OGeIRPOu2Q9HC-_ruZSX2oO9?usp=sharing#scrollTo=Dq3kGC3qktIN

[2] Ignacio Bugeño, Alfonso Ehijo, Felipe Gómez, Camila Rapu, "Ayudantía 5 – Introducción a TensorFlow", Escuela de Ingeniería, Universidad de O'Higgins. Reference: https://colab.research.google.com/drive/1kfswz0LjTnaTOePN4VFjq_TE1rDMuSkD?usp=sharing#scrollTo=MFPVwf9sjrUs

[3] Keras, Conv2D layer, Keras documentation. Reference: https://keras.io/api/layers/convolution_layers/convolution2d/

[4] Keras, Flatten layer, Keras documentation. Reference: https://keras.io/api/layers/reshaping_layers/flatten/