Tarea N° #3

Redes Neuronales Convolucionales

Luis Albandoz González

COM4402 – Introducción a Inteligencia Artificial

Escuela de Ingeniería, Universidad de O'Higgins

11 de noviembre, 2023

***Resumen*—**

El objetivo es implementar una Red Neuronal Convolucional que logre clasificar las imágenes correspondientes a la base de datos manuscritos MNIST y la base de datos de imágenes pequeñas CIFAR-10. Para ello se implementa una función de normalización de imágenes, definir el conjunto de entrenamiento y de prueba, también se implementan las funciones One-Hot Encording, CNN, una función CNN sin max pooling que ayuda a reducir las dimensiones.

Palabras claves— Red Neuronal Convolucional, imágenes, función de normalización, max pooling, conjunto entrenamiento, conjunto de prueba.

I. INTRODUCCIÓN

Una Red Neuronal Convolucional (CNN) es una clase de Red Neuronal en el aprendizaje profundo, aplicando con mayor frecuencia en el análisis de imágenes visuales. También se conoce como redes neuronales artificiales invariantes al desplazamiento o espacio, basada en la arquitectura de los pesos compartidos de los filtros de convolución que se deslizan a lo largo de las características de entrada proporcionando respuestas equivariantes a la traslación, conocida como feature maps. Tiene aplicaciones, principalmente en el reconocimiento de imágenes, clasificación de imágenes, análisis y procesamiento de imágenes.

II. MARCO TEÓRICO

A. *Redes Convolucionales*

Según Sotaquirá (2019) señala que "al igual que en las Redes Neuronales, las Redes Convolucionales también permiten detectar patrones en los datos de entrada, con la única diferencia de que en el caso de las Redes Convolucionales los datos de entrada son imágenes" (s.p). A partir de la afirmación anterior se puede mostrar que las Redes Convolucionales logran el procesamiento de imágenes en las diferentes capas de la neurona, lo que transforman a valores numéricos.

B. Convolución

Según Rapu (2023) una convolución "consiste en deslizar un pequeño filtro o kernel sobre la imagen de entrada, calculando el producto escalar entre estos, para construir un mapa de características" (p.3). De esta expresión se puede afirmar que la convolución durante el procesamiento de la imagen permite obtener un

valor escalar en el cual se almacena en un mapa, lo que permite obtener información relevante; por lo que el paso posterior a este será el procesamiento a las siguientes capas de la neurona.

C. *Pooling*

Según Rapu (2023) pooling "consiste en tomar grupos de pixeles en una región y calcular el valor representativo, como el máximo (Max Pooling) o el promedio (Average Pooling)" (p.4). A partir de lo anterior, se puede deducir que el pooling es una técnica que permite reducir la dimensionalidad de los datos, lo que permite reducir el costo computacional manteniendo información importante.

D. *Creación de una Red Neuronal Convolucional CNN*

Según Parada (2022) establece que "las capas convolucionales aplican distintos filtros sobre una imagen de entrada y crean nuevos volúmenes, por lo que las propiedades espaciales de la imagen se mantienen" (s.p). Con lo anterior se puede señalar, que las capas convolucionales pueden aplicar y modificar filtros en la imagen permitiendo mantener las propiedades iniciales de las imágenes antes de ser procesadas.

Parada (2022) afirma que "Lo más común es aplicar capas convolucionales seguidas de funciones de activación llamadas ReLU" (s.p). Lo anterior señala que recomendable utilizar esta función ya que, las capas convolucionales suelen aprovechar una función de activación *softmax* para clasificar los datos de manera adecuada, es decir, utiliza valores binarios 0 y 1.

E. *Optimizadores*

Según Rapu (2023) menciona que "su función principal es ajustar los pesos y sesgos de la red para minimizar la función de pérdida" (p.4). A partir de la afirmación se puede deducir que los optimizadores permiten mejorar la capacidad del modelo de una Red Convolucional logrando obtener mediciones más precisas. Los optimizadores que se utilizan más comúnmente son: Adam, Adagrad y Adadelta.

III. METODOLOGÍAS

Al abrir el código "*Tarea_3_IA_Luis_Albandoz.ipynb*" en Google Colab se debe elegir el entorno de ejecución GPU. La primera parte consiste en implementar una Red Convolucional (CNN) para MNIST a través de 7 pasos; y la segunda una implementación de la Red Neuronal Convolucional para CIFAR 10 a través de 4 pasos.

En las primeras 2 celdas del código se debe correr por separado, por lo que será el esqueleto de la implementación de las Redes Neuronales

Convolucionales mencionadas mostradas en la Fig. 1 y la Fig. 2:

```
"""Esta area configura el entorno Jupyter.
Por favor, no modifique nada en esta celda.
"""
import os
import sys
import time

# Importar módulos diversos
from IPython.core.display import display, HTML
```

Fig. 1 Código entorno Jupyter.

```
def _print_success_message():
    print('Pruebas superadas.')
    print('Puede pasar a la siguiente tarea.')

def test_normalize_images(function):
    test_numbers = np.array([0,127,255])
    OUT = function(test_numbers)
    test_shape = test_numbers.shape
```

Fig. 2 Pseudocódigo entorno función normalizar imágenes.

Realizando la Red Neuronal Convolucional a través de la base de datos MNIST, donde hay una muestra de 70000 vectores, los cuales 60000 pertenecen al conjunto de entrenamiento y 10000 en el conjunto de prueba.

En el paso 1 se implementa una función que normaliza las imágenes de 8 bits en el intervalo [0, 255] a [0, 1] como aparece en la Fig. 3:

```
def normalize_images(images):
    """Normalizar las imágenes de entrada.
    """
    # Normalizar la imagen aquí
    normalize_images = images/255
    images = normalize_images

    return images
```

Fig. 3 Función de normalización de la imagen.

En el paso 2 se debe añadir una nueva dimensión de entrada para las variables "*x_train*" y "*x_test*". Para la base de datos MNIST presenta una matriz de tamaño (28x28), y se transforma en (28x28x1) como se muestra en la Fig. 4

```
# Escribe aquí tu código
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
```

Fig. 4 Código expansión dimensiones.

En el siguiente paso se debe implementar la función para One – Hot Encording, lo cual codifica el vector de números en una matriz de K clases, como se muestra la definición que se muestra en la Fig. 5

```
def one_hot(vector, number_classes):
    """Devuelve una matriz codificada one-hot dado el vector argumento.
    """
    # Aquí almacenaremos nuestros one-hots
    one_hot = []

    # Aquí se codifica el 'vector' one-hot
    for i in vector:
      encording = [0] * number_classes
      encording[i] = 1
      one_hot.append(encording)

    # Transformar la lista en una matriz numpy y retornarla
    return np.array(one_hot)
```

Fig. 5 Código función one – hot encording.

En los pasos 4 y 5 se construye una Red Neuronal con convoluciones y se utiliza Max Pooling. En el paso 5 se definen los hiperparámetros hasta que "loss" y "val_loss" deben converger hasta valores bajos. En la Fig. 6 se aplica la función "net_1()" que implementa la red especificada con anterioridad.

```
def net_1(sample_shape, nb_classes):
    # Defina la entrada de la red para que tenga la dimensión `sample_shape
    input_x = Input(shape=sample_shape)

    # Cree aquí las conexiones internas de la red
    x = Conv2D(32, (3, 3), padding='same', activation='relu')(input_x)
    x = Conv2D(64, (3, 3), padding='same', activation='relu')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
```

Fig. 6 Red Convolucional con Max Pooling.

Particularmente, se definen los hiperparámetros: batch_size = 128 (tamaño de batch) y epochs = 10 (número de épocas de entrenamiento), lo que aparece en la Fig. 7

```
# Defina los hiperparámetros
batch_size = 128
epochs = 10
```

Fig. 7 Definición de hiperparámetros.

En el paso 6 se implementa una Red Neuronal Convolucional sin utilizar Max Pooling, en otras palabras, se implementa la función *net_2()*. En la Fig. 8 se muestra la implementación de la Red Convolucional sin Max Pooling, en el segundo bloque de convolución se añade un *stride* = 2 removiendo la capa de Max Pooling.

```
def net_2(sample_shape, nb_classes):
    # Defina la entrada de la red para que tenga la dimensión `sample_
    input_x = Input(shape=sample_shape)

    # Cree aquí las conexiones internas de la red
    x = Conv2D(32, (3, 3), padding='same', activation='relu')(input_x)
    x = Conv2D(64, (3, 3), padding='same', activation='relu', strides=
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
```

Fig. 8 Red Convolucional sin Max Pooling.

En el paso 7 se definen los hiperparámetros batch_size y epochs, los cuales se utilizan los mismos valores que aparecen en la Fig. 7

Posterior a ello se implementa otra Red Neuronal Convolucional para la base de datos CIFAR que consta de 60000 imágenes de color en 32x32 compuesta por 10 clases. Hay 50000 imágenes para el conjunto de entrenamiento y 10000 para el conjunto de prueba.

En el paso 8 se debe codificar la función one-hot encording creada anteriormente para codificar "*y_train*" y "*y_test*" como aparece en la Fig. 9

```
y_train = np.squeeze(y_train)
y_train = one_hot(y_train, 10)
y_test = np.squeeze(y_test)
y_test = one_hot(y_test, 10)
```

Fig. 9 Codificación one-hot para labels.

En el paso 9 se deben normalizar las imágenes de la base de datos CIFAR, se utiliza la función *normalize_images()* como aparece en la Fig. 3.

En el paso 10 se implementa una Red Neuronal Convolucional para la base de datos CIFAR. Para este caso en particular se utiliza la función *net_1()* lo cual se construye esta red utilizando Max Pooling, y se utiliza "*Adam*" como optimizador en que se muestra en la Fig. 10

```
# Dimensionalidad de la muestra
sample_shape = x_train[0].shape

# Construcción de la red
model = net_1(sample_shape, 10)
model.summary()

# Necesitamos compilar nuestro modelo de red neuronal
model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
```

Fig. 10 CNN para CIFAR

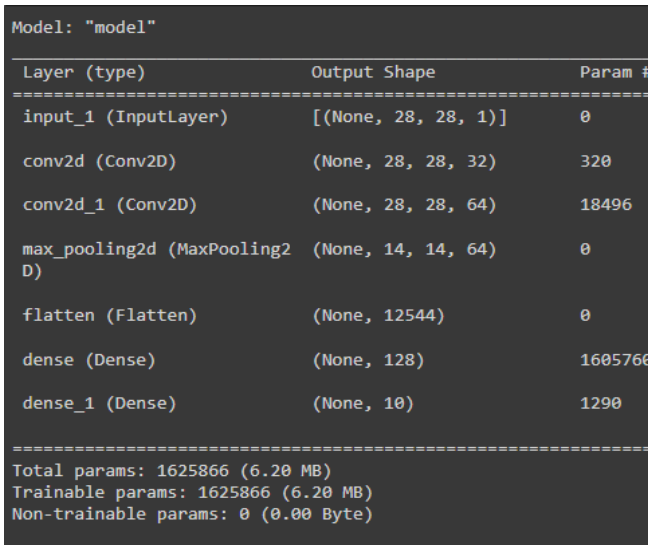
Y finalizando en el paso 11 se entrena este modelo o Red Neuronal, en el cual se entrena la red utilizando *epochs* = 30; *batch_size* = 128; *validation_split* = 0.2 utilizando la función *fit()* en el cual se muestra en la Fig. 11

```
# Construya el código dentro de esta celda
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=30,
                    validation_split=0.2)
```

Fig. 11 Entrenamiento CNN

IV. ANÁLISIS Y RESULTADOS OBTENIDOS

Al evaluar las Redes Neuronales Convolucionales de la base de datos MNIST (con Max Pooling y sin Max Pooling) y CIFAR, al entrenar se obtienen los siguientes resultados:



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d (Conv2D)             (None, 28, 28, 32)        320

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d (MaxPooling2  (None, 14, 14, 64)        0
 D)

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 128)               1605760

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1625866 (6.20 MB)
Trainable params: 1625866 (6.20 MB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. *12 Red Convolucional con Max Pooling MNIST*

Como se muestra en la Fig. 12 se analiza la Red Convolucional con Max Pooling utilizando la base de datos MNIST. A partir de esta figura se proporciona información sobre cada capa de la red, el tamaño de la capa y el número de parámetros entrenados en cada capa:

input_1 (InputLayer) no hay un parámetro entrenado debido a que esta es la capa de entrada.

conv2d (Conv2D) se observan 320 parámetros entrenados (32 filtros de 3x3 con un sesgo para cada filtro).

conv2d_1 (Conv2D) hay 18496 parámetros entrenados (64 filtros de 3x3 con un sesgo para cada filtro).

max_pooling2d (MaxPooling2D) no tiene parámetros entrenables, ya que es una capa Max Pooling sin pesos.

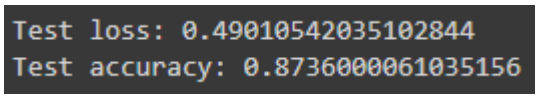
flatten (Flatten) no presenta parámetros entrenables, ya que es una capa de aplanamiento sin pesos.

dense (Dense) hay 1605760 parámetros entrenables (128 neuronas con 12544 entradas cada una, más 128 sesgos).

dense_1 (Dense) se entrenaron 1290 parámetros (10 neuronas con 128 entradas cada una, más 10 sesgos).

Los parámetros entrenados en total fueron 1625866.

En la Fig. 13 me muestra el resultado del modelo entrenado utilizando el conjunto de prueba (Red Convolucional con Max Pooling)

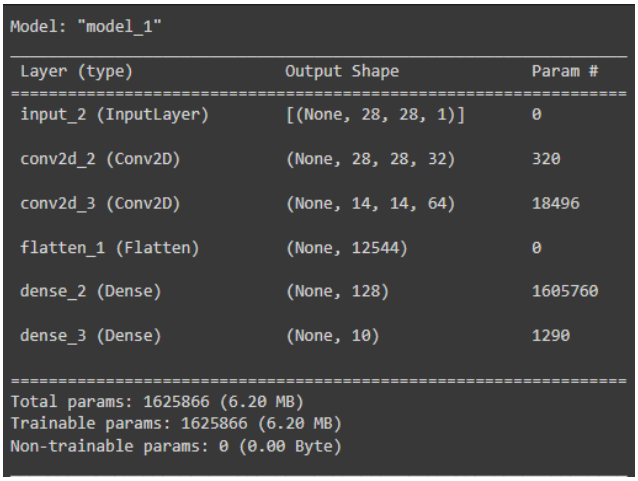


```
Test loss: 0.49010542035102844
Test accuracy: 0.8736000061035156
```

Fig. 13 Resultado Red Convolucional con Max Pooling MNIST

Se puede deducir que la evaluación del modelo en el conjunto de prueba indica que el modelo tiene una buena exactitud en la medición de los datos.

También se analiza la Red Convolucional, pero en ausencia de Max Pooling usando la base de datos MNIST, como se muestra en la Fig. 14



```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_2 (Conv2D)           (None, 28, 28, 32)        320

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        18496

 flatten_1 (Flatten)         (None, 12544)             0

 dense_2 (Dense)             (None, 128)               1605760

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1625866 (6.20 MB)
Trainable params: 1625866 (6.20 MB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. 14 Red Convolucional sin Max Pooling MNIST

En la Fig. 14 se muestra información sobre cada capa de la red, el tamaño y el número de parámetros en cada capa. En la

Fig. 12 y en la Fig. 14 se puede observar que tanto número de parámetros entrenados en total como la cantidad de parámetros por capa son iguales.

La Fig. 15 muestra el resultado del modelo entrenado utilizando el conjunto de prueba (Red Convolucional sin Max Pooling)

```
Test loss: 0.5062026381492615
Test accuracy: 0.8734999895095825
```

Fig. 15 Resultado Red Convolucional sin Max Pooling MNIST

El resultado obtenido por lo general es bastante bueno debido a la alta exactitud (accuracy), por lo que el rendimiento en el conjunto de prueba es bastante alto.

Analizando el modelo de Red Convolucional utilizando la base de datos CIFAR se utiliza la función *net_1()*, por lo que este modelo se utiliza con Max Pooling y se compila este modelo usando el optimizador "*Adam*" donde se muestra en la Fig. 11. Posteriormente se analiza este modelo de Red Convolucional, como se observa en la Fig. 16

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_4 (Conv2D)           (None, 32, 32, 32)        896

 conv2d_5 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 flatten_2 (Flatten)         (None, 16384)             0

 dense_4 (Dense)             (None, 128)               2097280

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 2117962 (8.08 MB)
Trainable params: 2117962 (8.08 MB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. 16 Red Convolucional con Max Pooling CIFAR y optimizador "Adam".

Al analizar este modelo se observa una capa de entrada de la forma (32,32,3) indicando imágenes de color con un tamaño de 32x32 pixeles y 3 canales RGB. Se utilizan 2 capas convolucionales con filtros de tamaño 3x3; la primera capa tiene 32 filtros y la segunda capa 64 filtros. Después de la segunda capa convolucional se aplica una capa Max Pooling de tamaño 2x2 reduciendo las dimensiones espaciales de la salida de la capa convolucional anterior. Después de la capa de Max Pooling se aplica una capa de aplanamiento (Flatten) para convertir la salida tridimensional de una forma unidimensional. Posterior a ello hay dos capas densas que están totalmente conectadas siguen al aplanamiento, con 128 y 10 neuronas respectivamente. La última capa tiene 10 neuronas, que corresponden a las 10 clases en el conjunto de datos CIFAR – 10. La cantidad total de parámetros es de 2117962, y todos son entrenables.

Posterior a ello se entrena este modelo con los parámetros que se definen en la Fig. 11, utilizando el conjunto de prueba de la base de datos CIFAR – 10, a continuación, se muestra el resultado del entrenamiento de esta red convolucional a partir de la Fig. 17

```
Test loss: 2.8946518898010254
Test accuracy: 0.6596999764442444
```

Fig. 17 Resultado Red Convolucional con Max Pooling CIFAR y optimizador "Adam".

Analizando la Red Convolucional con Max Pooling utilizando la base CIFAR, con optimizador "*Adagrad*". Para ello se debe cambiar el parámetro *optimizer* que aparece en la Fig. 10. Analizando este modelo de Red Convolucional se observan los mismos parámetros de la Fig.16, y el resultado del entrenamiento para esta Red Convolucional utilizando el conjunto de prueba se muestra en la Fig. 18

```
Test loss: 1.4303771257400513
Test accuracy: 0.49140000343322754
```

Fig. 18 Resultado Red Convolucional con Max Pooling CIFAR y optimizador "Adagrad".

Por su parte, en la Red Convolucional con Max Pooling utilizando la base CIFAR, con optimizador "*Adadelta*". Para ello se debe cambiar el parámetro *optimizer* que aparece en la Fig. 10. Observando este modelo de Red Convolucional se observan los mismos parámetros de la Fig.16, y el resultado del entrenamiento para esta Red Convolucional utilizando el conjunto de prueba se muestra en la Fig. 19

```
Test loss: 1.9029603004455566
Test accuracy: 0.3555999994277954
```

Fig. 19 Resultado Red Convolucional con Max Pooling CIFAR y optimizador "Adadelta".

Al comparar los modelos anteriores a través de los resultados del entrenamiento utilizando el conjunto de prueba, como se muestran en las siguientes figuras: Fig. 13, Fig. 15, Fig. 17, Fig. 18 y Fig. 19 se pueden observar que el primer modelo de la Red Convolucional (Red Convolucional con Max Pooling MNIST) presenta un mejor funcionamiento debido a que presenta un menor valor de perdida (*Test loss*) de un 0.4901 y comparando el accuracy para el conjunto de prueba (*Test accuracy*) de un 0.8736 (87.36%) presenta una mejor exactitud en la clasificación de imágenes.

Al analizar los optimizadores que se utilizaron para el modelo de Red Convolucional con la base de datos CIFAR – 10 utilizando los optimizadores: Adam, Adagrad y Adadelta; al entrenar este modelo utilizando estos optimizadores, observando en la Fig. 17, Fig. 18 y Fig. 19 se observa que al utilizar el optimizador Adagrad hay una menor perdida con un *Test loss* con 1.43; pero en cuanto al accuracy, utilizando el optimizador Adam clasifica con mayor exactitud las imágenes, presentando un accuracy de un 0.6597 (65.97%). En términos de menor pérdida y mayor precisión en el conjunto de prueba, el optimizador Adagrad presenta un mejor funcionamiento.

Observando el overfitting para los modelos de Redes Convolucionales, en los modelos que se utiliza la base de datos MNIST (con Max Pooling y sin Max Pooling), dado que el valor de perdida (*Test loss*) es ligeramente más alto que la precisión, como se observan en las Fig. 13 y Fig. 15, se podría deducir que no se experimenta un overfitting significativo. Sin embargo, en el modelo de Red Convolucional utilizando la base CIFAR – 10, como se observan en las Fig. 17, Fig. 18 y Fig. 19 que el valor de perdida es extremadamente alto si se compara con el valor de la precisión; en este aspecto se puede afirmar que hay un sobreajuste (overfitting) en cuanto a la medición de las imágenes.

Para perfeccionar el rendimiento pueden tener varias estrategias para mejorar el modelo, incluyendo el ajuste de hiperparámetros, la exploración de diferentes arquitecturas de red, la aplicación de técnicas de regularización, la implementación de aumento de datos y la realización de un análisis más detallado de los errores del modelo para identificar posibles áreas de mejora. Utilizando estas estrategias se podría optimizar el rendimiento para la Red Convolucional CIFAR.

V. CONCLUSIONES GENERALES

Una Red Convolucional es un tipo de Red Neuronal que permite realizar procesamiento de imágenes y segmentar objetos a través de las diferentes capas de la neurona (capas convolucionales). Una de las aplicaciones más relevantes en la Red

Convolucional es la técnica del Pooling o capa de agrupación que consiste en reducir la dimensionalidad de los datos manteniendo información importante permitiendo el ahorro computacional; por ejemplo: al utilizar Max Pooling con la dimensionalidad reducida se elige el valor máximo.

Al examinar los distintos modelos de Redes Convolucionales, utilizando la base de datos MNIST, con Max Pooling y sin Max Pooling se puede concluir que no hay un overfitting significativo debido a que el valor de pérdida es levemente mayor que el valor del accuracy durante el entrenamiento con los datos del conjunto de prueba, los hiperparámetros que se utilizan para estos modelos de Redes Convolucionales son el tamaño de batch (*batch_size* = 128) y el número de épocas (*epochs* = 10). Analizando los valores de *Test_loss* y *Test_accuracy* se puede concluir que la Red Convolucional con Max Pooling presenta una menor pérdida en la medición de los datos y clasifica mejor las imágenes.

Usando el modelo de Red Convolucional con Max Pooling, trabajando con la base de datos CIFAR – 10, a diferencia de otros modelos de Red Convolucional se observa un mayor sobreajuste u overfitting en la medición de las imágenes debido a que el valor de pérdida es significativamente mayor que el valor del accuracy realizando la medición del conjunto de prueba. Se trabajan con los optimizadores *Adam*, *Adagrad* y *Adadelta* para analizar los valores de pérdida (*Test loss*) y del accuracy (*Test accuracy*). Para esta Red Convolucional, al comparar los valores de *test loss* y *test accuracy* se recomienda utilizar el optimizador *Adagrad* ya que presenta un menor valor de pérdida y una mayor efectividad en la clasificación de imágenes.

REFERENCIAS BIBLIOGRÁFICAS

[1] Sotaquirá, M. (2019). ¿Qué son las Redes Convolucionales?. Codificandobits. https://www.codificandobits.com/blog/redes-convolucionales-introduccion/

[2] Rapu, C. (2023). AyudantIA 5: Redes neuronales convolucionales (CNNs) e introducción a Tensor Flow. Universidad de O'Higgins. [PDF]

[3] Parada, P. (2022). Qué son las Redes Neuronales Convolucionales. IEBS. https://www.iebschool.com/blog/redes-neuronales-convolucionales-big-data/

[4] Rapu, C (2023). AyudantIA 6: Redes neuronales con TensorFlow. Universidad de O'Higgins. [PDF]