Tarea N°3:

Redes Neuronales Convolucionales

Cristóbal Tomás David Alberto Lagos Valtierra
Escuela de Ingeniería, Universidad de O'Higgins
11 de Noviembre del 2023

***Abstract*—El objetivo de este informe es utilizar redes neuronales convolucionales para resolver problemas de clasificación de labels. Se aplicarán técnicas de entrenamiento de modelos a 2 dataset los cuales son la base de datos de manuscritos MNIST y la base de datos de imágenes pequeñas CIFAR-10. Para lograr esto, haremos uso de Python con la librería TensorFlow especializada en realizar cálculos numéricos y construir modelos de aprendizaje automático.**

Para poder realizar un correcto entrenamiento de los modelos, se aplicarán diversas técnicas de preprocesamiento de los datos como lo son la normalización de datos, ampliación de dimensionalidad en los datos de entrada y codificación One-Hot.

Una vez entrenados los modelos se mostrará al lector el impacto que puede resultar en las métricas de un modelo, la utilización de Max-Pooling como operación para reducir la dimensionalidad espacial de la representación. Esto con el objetivo de evidenciar si el uso de esta técnica resulta siempre en una mejora en el entrenamiento de modelos de redes neuronales convolucionales.

I. Introducción

El campo del aprendizaje profundo ha experimentado un avance significativo en los últimos años, transformando radicalmente la manera en que abordamos problemas complejos en diversas disciplinas. Entre las arquitecturas de redes neuronales que han demostrado un rendimiento excepcional en tareas relacionadas con la visión por computadora, se destacan las Redes Neuronales Convolucionales (CNN). Este informe se centra en explorar y analizar el uso de CNN, con un enfoque particular en la clasificación de datos provenientes de las bases de datos MNIST y CIFAR-10, así como mostrar el impacto que tiene el utilizar o no una de las técnicas con más éxito de Pooling, Max-Pooling.

Dada la complejidad inherente de las imágenes, la calidad de los resultados obtenidos por estos modelos depende en gran medida de cómo se manipulan y preparan los datos de entrada. Es por esto que, para realizar un adecuado entrenamiento de CNN, es de vital importancia realizar un correcto preprocesamiento de los datos, esto para que se puedan aprovechar de la mejor forma el uso de herramientas, ya que estas esperan los datos de entrada en un determinado formato. Un correcto preprocesamiento no solo garantiza una representación más eficiente de la información, sino que también facilita la convergencia durante el entrenamiento, mejorando la generalización del modelo y evitando problemas como lo es el Sobreajuste de los datos.

Respecto a las aplicaciones de las CNN podemos mencionar algunas que son vastamente utilizadas en la sociedad actual, como la detección de objetos y reconocimiento facial hasta la clasificación de patrones complejos en imágenes médicas. La capacidad de estas redes para aprender representaciones jerárquicas y encontrar características discriminativas las convierte en herramientas muy poderosas en el análisis de datos visuales. En este informe, nos enfocaremos en la clasificación, evaluando el rendimiento de una CNN en dos conjuntos de datos emblemáticos: MNIST, una colección de dígitos manuscritos, y CIFAR-10, una base de datos que desafía al modelo con imágenes en color de objetos en diversas categorías.

El objetivo principal de este informe es implementar Redes Neuronales Convolucionales capaces de lograr una clasificación precisa en las bases de datos mencionadas y como el Max-Pooling en los modelos resultantes. Lo anterior implica no solo la construcción de una arquitectura de red efectiva, sino también la selección y aplicación estratégica de técnicas de preprocesamiento de datos para optimizar el rendimiento del modelo. A medida que avanzamos en este informe, exploraremos detalladamente lo que implica el diseño de una red de este tipo, las estrategias de preprocesamiento implementadas y evaluaremos el rendimiento final en términos de precisión y pérdida en el conjunto de entrenamiento y validación. En última instancia, se espera que los resultados obtenidos denoten eficacia de las CNN y el uso de Max-Pooling en la clasificación de datos complejos y sirvan como referencia para futuras aplicaciones en campos que requieran análisis visual avanzado.

II. Marco Teórico

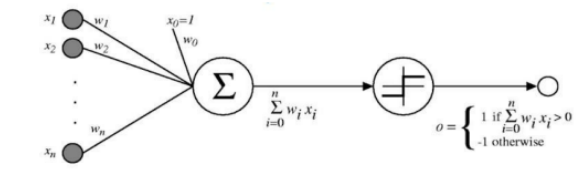
A. Machine Learning

Machine Learning es una rama de la inteligencia artificial que se enfoca principalmente en desarrollar algoritmos y modelos que permiten a las computadoras aprender y mejorar su desempeño en tareas específicas a través de la experiencia, sin la necesidad de ser programadas de manera explícita.

En particular, el Aprendizaje Supervisado permite que la máquina aprenda de manera explícita a partir de los datos proporcionados. Además de los datos de entrada, estos van acompañados de una etiqueta con la clase a la que corresponde, es decir, los datos de salida. A medida que va aprendiendo se le informa al modelo si realizó una correcta asignación a la clase esperada con el objetivo de ajustar los parámetros y mejorar en las métricas que evalúan que tan bien realizó una asignación.

B. Redes Neuronales

Las redes neuronales son un tipo de modelo aplicable al campo de aprendizaje supervisado inspirado en el funcionamiento del cerebro humano, en particular, a las redes de neuronas presentes en el cerebro. Este modelo está diseñado para procesar información de manera similar a como lo hacen las neuronas en el cerebro, esto lo hace implementando el modelo conocido como Perceptrón.



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Fig. 1: Neurona y perceptrón.

El perceptrón toma un conjunto de entradas (características) ponderadas y las suma. Luego, aplica una función de activación (generalmente una función escalón o similar) al resultado de la suma ponderada. Si el valor resultante cumple cierto umbral, la neurona se activa y produce una salida; de lo contrario, no produce ninguna salida.

Existen diversos tipos de redes neuronales, cada una enfocada con diversos propósitos para abordar de mejor forma diversas tareas, entre ellas encontramos la utilizada en este esta investigación, Redes neuronales del tipo Convolucionales.

C. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales (CNN por sus siglas en inglés, Convolutional Neural Networks) son un tipo de arquitectura de redes neuronales diseñadas especialmente para procesar datos estructurados en forma de matrices, como lo son las imágenes. Estas redes han demostrado un rendimiento excepcional en tareas de visión por computadora, como clasificación de imágenes, detección de objetos y segmentación semántica (algoritmo de deep learning que asocia una etiqueta o categoría a cada píxel presente en una imagen).

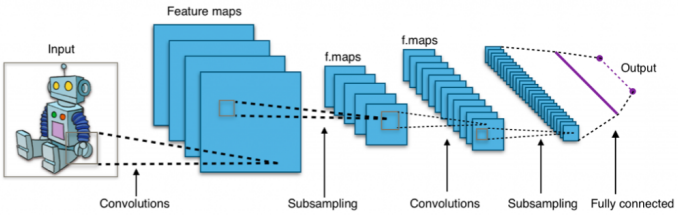


Fig. 2: Redes neuronales convolucionales.

Las CNN se inspiran en la organización y funcionamiento del sistema visual biológico, donde las neuronas individuales responden a regiones específicas del campo visual. A diferencia de las redes neuronales estándar, las CNN utilizan capas de convolución para realizar operaciones locales en regiones de la entrada, permitiendo así la detección de patrones y características importantes.

Un ejemplo que evidencia el porqué de lo anterior sería en el contexto de un clasificador que busca reconocer un determinado objeto en una imagen, es esencial lograr la capacidad de identificarlo sin importar su ubicación específica en la imagen. Dado este hecho, si calculamos ciertas estadísticas para una parte de la imagen de entrada, es probable que esas estadísticas también sean relevantes para otras áreas de la imagen. Este principio se explora en las redes neuronales convolucionales, que se diseñan para aprender características aplicables a diversas partes de una imagen, contribuyendo así a la eficacia de estas.

Para lograr lo anterior, es que se usa el concepto conocido como "ventanas deslizantes", que es el encargado de definir regiones de interés en una imagen, desplazándose para escanear toda la imagen. Al utilizar esta ventana deslizante para determinar la entrada visible para una pequeña red neuronal, se realiza lo que se conoce como convolución. Este enfoque contribuye a la capacidad de este tipo de redes para capturar patrones y características clave en imágenes.

Como resultado del uso de las ventanas deslizantes tenemos el mapa de características, que es la salida producida por la aplicación de filtros sobre una imagen de entrada. La convolución implica deslizar un conjunto de filtros (kernels) sobre la imagen y calcular el producto punto entre los valores del filtro y los píxeles correspondientes de la región actual de la imagen. Finalmente, este proceso es el principal responsable de captar gran variedad de características o rasgos dentro de una imagen.

Con el objetivo de controlar el tamaño del mapa de características podemos hacer uso de los conceptos conocidos como *padding* y *strides*:

1) Padding: se refiere a la adición de píxeles adicionales alrededor de los bordes de una imagen antes de aplicar una operación de convolución
2) Stride: indica la cantidad de píxeles que se desplaza la ventana de convolución a medida que se desliza a lo largo de la imagen de entrada.

En conjunto con los conceptos previamente mencionados, tenemos la operación de pooling, el cual es una técnica comúnmente utilizada para reducir la dimensionalidad espacial de los mapas de características y extracción de características clave. Hay varios tipos de pooling, los más comunes son:

1) Max-Pooling: En cada ventana, se selecciona el valor máximo y se utiliza como representación de esa región en el mapa de características resultante. Resulta particularmente efectivo en contextos que se busca preservar las características más destacadas.
2) Average-Pooling: En cada ventana, se calcula el promedio de los valores y se utiliza como representación de esa región en el mapa de características resultante. Una de las ventajas principales es que esta técnica es menos propensa al sobreajuste y proporciona una forma de resumen más suave de las características.
3) Min-Pooling: En cada ventana, se selecciona el valor mínimo y se utiliza como representación de esa región

en el mapa de características resultante. Sigue el mismo principio de resaltar características importantes.

D. Codificación one-hot

Técnica utilizada en procesamiento de datos y aprendizaje automático para representar categorías o clases de manera binaria. En este esquema, cada categoría se representa mediante un vector binario donde todos los elementos son cero, excepto uno, indicando la pertenencia a la categoría correspondiente.

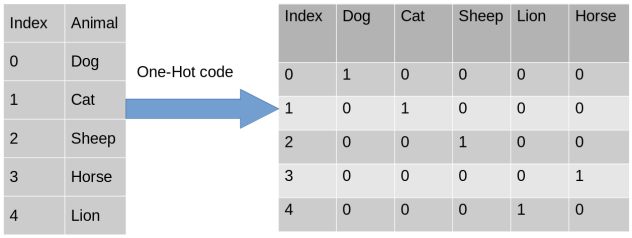


Fig. 3: Codificación one hot.

Este tipo de representación es especialmente útil cuando se trabaja con tareas de clasificación y se quiere evitar la asignación implícita de un orden o relación ordinal entre las categorías. Cada categoría es tratada como una entidad única e independiente.

Esta codificación es una representación ideal para entrenar un modelo mediante el algoritmo de descenso de gradiente con la función softmax.

E. Función Softmax

El algoritmo de descenso de gradiente para la función de activación softmax es un componente clave en la optimización de modelos de clasificación multiclase. Se utiliza comúnmente para convertir un vector de valores reales en una distribución de probabilidad, lo que es fundamental en problemas de clasificación con múltiples clases.

El objetivo es ajustar los pesos y sesgos para minimizar una función de pérdida (loss) asociada con la tarea de clasificación. La función de pérdida comúnmente utilizada en conjunto con softmax es la entropía cruzada (cross-entropy).

F. Ampliación de la dimensión de la entrada

La técnica de ampliar la dimensión de la entrada desempeña un papel esencial en la adaptación de datos para su integración en redes neuronales convolucionales, ya que permite trabajar con conceptos como canales de color y mapas de características. Este procedimiento implica la transformación de la representación original de una imagen, que podría presentarse como un arreglo bidimensional, hacia una forma tridimensional que ahora incluye la dimensión adicional del canal.

En términos más específicos, para imágenes en escala de grises, este proceso de expansión de la dimensión se traduce en cambiar de una matriz de tamaño (ancho, alto) a un tensor tridimensional de tamaño (ancho, alto, 1). Esta modificación es fundamental para garantizar que la red neuronal pueda interpretar eficientemente la información espacial y estructural de la imagen durante el proceso de entrenamiento.

Si bien uno puede penar que esta técnica no aporta mayor información, la inclusión de la dimensión adicional del canal ofrece ventajas significativas al permitir que la red capture y procese patrones más complejos presentes en los datos de entrada. Esta adaptación posibilita una representación más rica y detallada de la información visual, mejorando así la capacidad de la red para aprender y generalizar a partir de los datos de entrada.

G. Normalización de los datos

La normalización de datos es un componente esencial en el procesamiento previo de información, esta técnica consiste en ajustar los valores de los valores de entrada de un conjunto de datos para que sigan una escala común, generalmente una distribución centrada en cero con desviación estándar de uno.

Este procedimiento es crucial para garantizar que todas las características contribuyan equitativamente al proceso de entrenamiento de modelos, independientemente de sus escalas originales. En el contexto de redes neuronales, como las utilizadas en este estudio, la normalización es particularmente valiosa para acelerar la convergencia del modelo durante el entrenamiento.

La normalización de datos aborda problemas potenciales asociados con variaciones significativas en las magnitudes de las características, lo que puede afectar negativamente la eficiencia del modelo. Al aplicar la normalización, permitimos que los algoritmos de aprendizaje automático converjan de manera más rápida y efectiva. Este proceso permite evitar que las características con valores más grandes dominen la contribución al modelo, garantizando así una ponderación equitativa durante el entrenamiento.

III. Metodología

A. Características de los Datos

Las bases de datos a utilizar corresponde al dataset de manuscritos MNIST y la base de datos de imágenes pequeñas CIFAR-10.

La base de datos MNIST consiste en un conjunto de 70,000 imágenes de dígitos escritos a mano en escala de grises, cada una con dimensiones de 28x28 píxeles. Esta base de datos es ampliamente utilizada para tareas de clasificación de imágenes y representa un desafío significativo debido a la variabilidad en la escritura, la presencia de ruido y la simplicidad aparente de los datos. Al poseer estas características es un dataset ideal para que se puedan aplicar técnicas de convolución de manera que se puedan identificar rasgos y características dentro de una imagen. Cada imagen está etiquetada con la clase correspondiente del dígito que representa (0 a 9).

La base de datos CIFAR-10, por otro lado, presenta un conjunto más diverso de desafíos. Compuesto por 60,000 imágenes a color divididas en 10 clases (automóviles, aviones, pájaros, gatos, ciervos, perros, ranas, caballos, barcos y camiones), cada imagen tiene dimensiones de 32x32 píxeles. CIFAR-10 es conocida por su complejidad visual, variabilidad en la orientación y presencia de objetos en un contexto más

amplio. La diversidad de clases y la baja resolución de las imágenes hacen que la tarea de clasificación en CIFAR-10 sea un banco de pruebas robusto para evaluar el rendimiento de modelos de clasificación de imágenes.

Ambos conjuntos de datos se utilizarán para entrenar y evaluar modelos CNN con el objetivo de realizar la clasificación de imágenes. La elección de estas bases de datos se basa en su representatividad en problemas de clasificación de imágenes a diferentes escalas de complejidad.

B. Preprocesamiento de los datos

Se implementarán técnicas estándar de preprocesamiento de datos, como la normalización, aumento de dimensionalidad y una separación apropiada en conjunto de entrenamiento y validación. Lo anterior con el objetivo de mejorar la capacidad del modelo para generalizar y realizar predicciones precisas en datos no vistos.

Implementaremos funciones que nos permitan realizar lo anterior, para realizar los testeos que nos permitan comprobar el correcto funcionamiento de estas, haremos uso de la base de datos MNIST.

En primer lugar, implementaremos la función que realiza la normalización de datos:

```python
def normalize_images(images):
    """Normalizar las imágenes de entrada.
    """
    images = images/255

    return images
```

Fig. 4: Normalización.

Esta implementación resulta bastante sencilla, ya que sabemos que las matrices correspondientes a las imagenes poseen valores en un rango [0,255], por lo que si dividimos cada uno de estos valores por 255, logramos la normalización de datos a un intervalo de [0,1].

Ahora procederemos a ampliar la dimensión de la entrada, para ello haremos uso de la función *numpy.expand_dims()* de la librería numpy:

```python
# Agregar una nueva dimensión a x_train y x_test
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
```

Fig. 5: Ampliar dimensión de entrada.

Lo anterior corresponde a transformar los datos de entrada añadiendo una dimensión con el objetivo de tener los datos en un formato adecuado para el entrenamiento de las CNN.

A continuación, procederemos a transformar los datos de salida aplicando la técnica One-Hot:

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

    # Para cada muestra en el vector
    for sample in vector:
        # Crear un array con ceros de tamaño number_classes
        one_hot_sample = np.zeros(number_classes)
        # Establecer un 1 en la posición correspondiente a la muestra actual
        one_hot_sample[sample] = 1
        # Agregar el one-hot de la muestra a la lista
        one_hot.append(one_hot_sample)

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

Fig. 6: One-Hot.

Los pasos implementados en la codificación One-Hot fueron:

1) Recibir como argumentos un vector con N muestras (dimensiones) y un número K correspondiente al número de clases.
2) Para cada muestra del vector, crear un array con K dimensiones.
3) En el array añadir ceros en todas las posiciones excepto en la posición indicada por la muestra actual en el vector de entrada.

C. Implementación CNN

Teniendo listo el apartado del preprocesamiento, procederemos a definir una función que implementará nuestra primera Red Neuronal Convolucional (net_1):

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

    # Generar 32 kernel maps utilizando una capa convolucional
    x = Conv2D(32, (3, 3), activation='relu', strides=(1, 1) , padding='same')(input_x)

    # Generar 64 kernel maps utilizando una capa convolucional
    x = Conv2D(64, (3, 3), activation='relu', strides=(1, 1) , padding='same')(x)

    # Reducir los feature maps utilizando max-pooling
    x = MaxPooling2D(pool_size=(2, 2), strides=(1, 1) , padding='same')(x)

    # Aplanar el feature map
    x = Flatten()(x)

    # Capa fully-connected a 128 dimensiones
    x = Dense(128, activation='relu')(x)

    # Capa fully-connected a `nb_classes` dimensiones
    probabilities = Dense(nb_classes, activation='softmax')(x)

    # Definir la salida
    model = Model(inputs=input_x, outputs=probabilities)

    return model
```

Fig. 7: Primera red neuronal convolucional (net_1).

Los pasos implementados en esta función son:

1) Definir una entrada.
2) Generar 32 kernel maps utilizando una capa convolucional.
3) Generar 64 kernel maps utilizando una capa convolucional.

4) Reducir los feature maps utilizando max-pooling.
5) Aplanar el feature map.
6) Capa fully-connected, usando Dense, a 128 dimensiones.
7) Capa fully-connected, usando Dense, a K clases (argumento) dimensiones.

En conjunto con la función anterior, implementaremos una segunda función (net_2) que realizará entrenamiento de CNN:

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

    # Generar 32 kernel maps utilizando una capa convolucional (stride=1)
    x = Conv2D(32, (3, 3), activation='relu', strides=(1, 1) , padding='same')(input_x)

    # Generar 64 kernel maps utilizando una capa convolucional (stride=2)
    x = Conv2D(64, (3, 3), activation='relu', strides=(2, 2),  padding='same')(x)

    # Aplanar el feature map
    x = Flatten()(x)

    # Capa fully-connected a 128 dimensiones
    x = Dense(128, activation='relu')(x)

    # Capa fully-connected a `nb_classes` dimensiones
    probabilities = Dense(nb_classes, activation='softmax')(x)

    # Definir la salida
    model = Model(inputs=input_x, outputs=probabilities)

    return model
```

Fig. 8: Segunda red neuronal convolucional (net_2).

La motivación principal de crear esta función es añadir 2 diferencias clave, las cuales son:

1) Remover la capa de Max-Pooling.
2) Añadir un stride = 2 en la segunda capa de convolución.

Más adelante profundizaremos más a fondo el porqué de la creación de esta segunda función que remueve Max-Pooling.

IV. Resultados

Se realizó el entrenamiento de las redes neuronales en el conjunto de datos MNIST haciendo uso net_1 utilizando como hiper-parámetros un *batch_size* = 128 y un número de *epochs* = 16 obteniendo el siguiente gráfico:

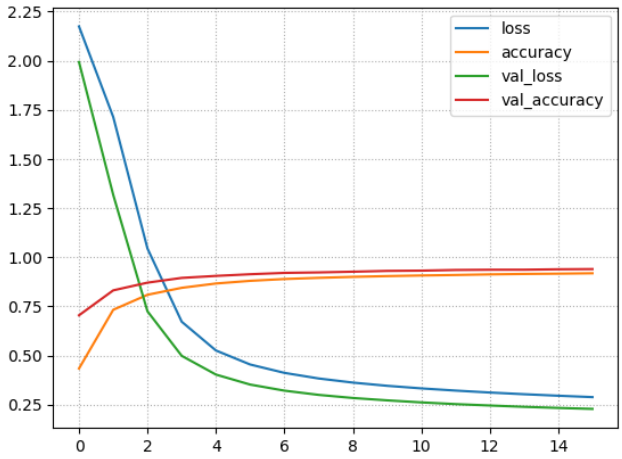


Fig. 9: Aplicación net_1 a MNIST.

A su vez, al mismo conjunto de datos procederemos a aplicar nuestra función (net_2), que recordemos que tiene como principal diferencia el que no incluye Max-Pooling, así como un uso de *stride* = 2:

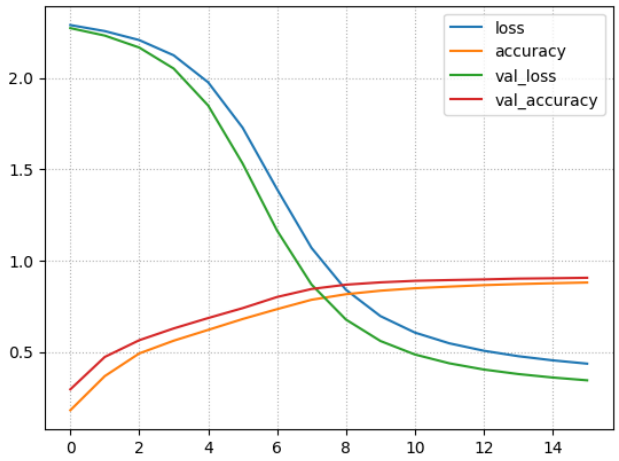


Fig. 10: Aplicación net_2 a MNIST.

A continuación graficaremos aplicación de net_1 en la base de datos CIFAR-10 con la misma elección de hiper-parámetros, esto con el fin de observar como se comporta el entrenamiento del modelo en un mismo número de épocas que el utilizado en MNIST:

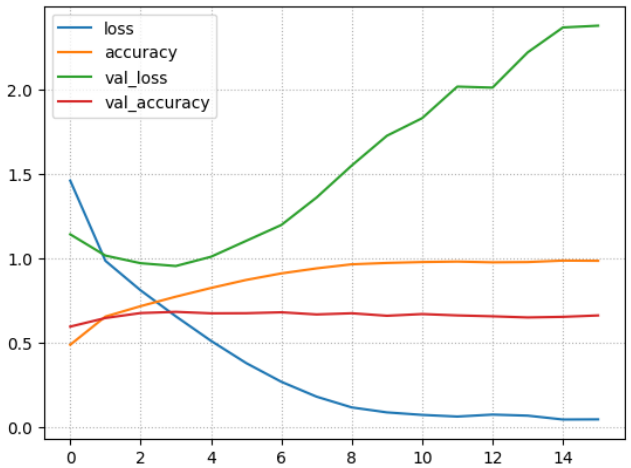


Fig. 11: Aplicación net_1 a CFIR-10.

Por último, procedemos a aplicar la función net_2:

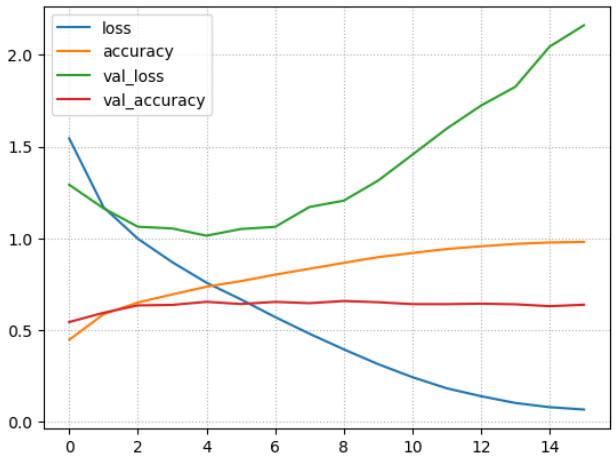


Fig. 12: Aplicación net_2 a CFIR-10.

En los ejemplos anteriores se usó como optimizador a Adam, puesto que fue el que resultó con mejores resultados, de igual forma, a continuación se muestra el resultado de utilizar otros optimizadores usando net_2:

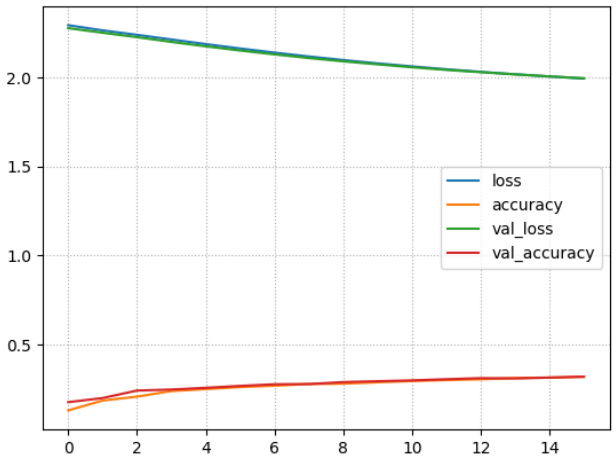


Fig. 13: Aplicación net_2 a CFIR-10 (Adadelta).

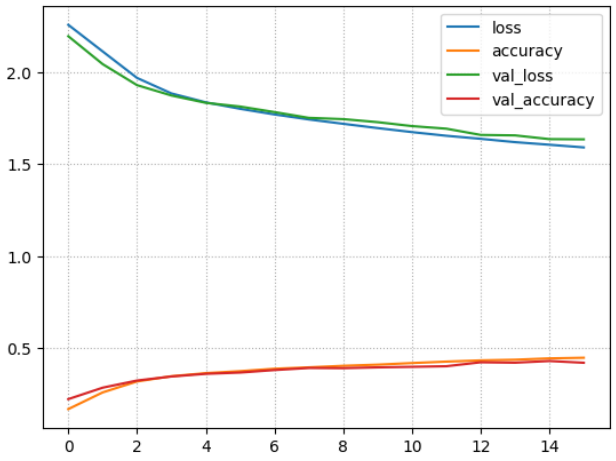


Fig. 14: Aplicación net_2 a CFIR-10 (Adagrad).

V. Análisis

En cuanto al uso o no de Max-Pooling, se evidencia con mayor fuerza en el dataset MINST que al incluir esta técnica toma un menor número de épocas en llegar a una convergencia en las distintas métricas que se utilizan para medir el rendimiento de los modelos.

Al analizar los gráficos resultantes de la medición de las métricas accuracy y loss en los distintos son varios los comportamientos que son interesantes de mencionar.

En primer lugar, el modelo que resultó con mejores valores de accuracy fue net_1 usando como optimizador a Adam, llegando a un mejor número en accuracy con un máximo de 0.9352 en el conjunto de validación en el conjunto de datos MNIST. Si bien, en CIFAR-10 obtuvo un valor de un 0.9865 en accuracy, al ver esta métrica en el conjunto de validación podemos observar que obtuve un rendimiento de solamente un 0.6531. Además, si nos fijamos en el loss del conjunto de validación podemos ver claramente que hay un punto en que la pérdida es mínima, pero luego de la época 4 este valor se dispara llegando a valores máximos de esta métrica. Por lo anterior, podemos estar seguros que está ocurriendo un Sobreajuste de los datos al utilizar 16 épocas.

Por otra parte, al indagar qué optimizador resulta mejor podemos decir con seguridad que es Adam, esto debido a que al utilizar optimizadores como Adadelta y Adagrad, al ver los gráficos resultantes de estos podemos observar que el accuracy de ambos parece converger a valores cercanos e inferiores a 0.5, comportamiento similar al ver los valores loss resultantes.

Con el objetivo de mejorar aún más el rendimiento, podríamos implementar distintas estrategias que soluciones distintos problemas, tales como agregar early stopping para evitar Sobreajuste, revisar un mayor análisis en los datos de entrada y salida correspondientes al dataset CFIR-10, ya que difiere bastante en las métricas en comparación a MNIST.

VI. Conclusiones Generales

En el transcurso de este informe, hemos indagado en diversas partes clave en el desarrollo y entrenamiento de modelos de redes neuronales para tareas de clasificación de imágenes. Una de las conclusiones más destacadas es la importancia crítica de emplear distintos optimizadores durante el proceso de entrenamiento. La elección del optimizador puede impactar significativamente en la velocidad de convergencia y la capacidad del modelo para generalizar a nuevos datos.

Asimismo, hemos resaltado la relevancia de buscar el modelo más adecuado para la tarea en cuestión. La arquitectura de la red y la configuración de sus capas son elementos cruciales que influyen en el rendimiento del modelo. La experimentación del uso o no de Max-Pooling como técnica permitió evidenciar la importancia de esta técnica en la velocidad de convergencia del entrenamiento de un modelo.

Detectar el Sobreajuste también ha emergido como un punto clave de consideración. La capacidad de los modelos para ajustarse demasiado a los datos de entrenamiento puede comprometer su habilidad para generalizar a nuevos datos.

En conjunto, estas conclusiones refuerzan la idea de que la elección cuidadosa de optimizadores, la búsqueda del modelo

más apropiado, la atención al Sobreajuste y la utilización de redes neuronales convolucionales son pasos fundamentales para el desarrollo exitoso de modelos de aprendizaje automático en tareas de visión por computadora. La combinación de estos elementos no solo mejora la precisión de las predicciones, sino que también contribuye a la creación de modelos más robustos y adaptables a una variedad de escenarios y conjuntos de datos.

VII. Bibliografía

1) https://www.tensorflow.org/?hl=es-419
2) https://www.ibm.com/es-es/topics/convolutional-neural-networks
3) https://la.mathworks.com/solutions/image-video-processing/semantic-segmentation.html
4) https://datasmarts.net/es/la-historia-del-conjunto-de-datos-mas-popular-de-computer-vision-mnist/#::text=MNIST%20significa%20Modified%20NIST%2C
5) https://datasmarts.net/es/que-es-cifar-10/
6) https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/