Tarea N°3:

Redes Neuronales Convolucionales

Cesar Hormazabal Galleguillos
Escuela de Ingeniería, Universidad de O'Higgins
11, Noviembre, 2023

***Abstract*—Como objetivo se tiene el utilizar distintos algoritmos que implementan la arquitectura de redes neuronales convolucionales. Se utilizo en este informa la base de datos "Mnist" a la cual se le implementan 3 modelos distintos, para comparar los resultados con el fin de contestar las interrogantes asociadas y ademas concluir que variantes son mejores.**

I. Introducción

El objetivo de esta tarea es implementar una Red Neuronal Convolucional que logre clasificar los labels correspondientes a la base de datos de manuscritos MNIST y la base de datos de imágenes pequeñas CIFAR-10.
La tarea fue realizada en jupyter notebook Python usando TensorFlow.

II. Marco Teórico

Las Redes Neuronales Convolucionales (CNN) han demostrado ser poderosas herramientas en el campo del aprendizaje profundo, especialmente para tareas relacionadas con el procesamiento de imágenes. Estas redes están diseñadas para reconocer patrones y características espaciales en datos visuales, imitando en cierto sentido la forma en que el cerebro humano procesa información visual.

A. Arquitectura de una CNN

Una CNN típica consta de capas convolucionales, capas de agrupación (pooling) y capas completamente conectadas. Las capas convolucionales aplican filtros a regiones locales de la entrada para extraer características, mientras que las capas de agrupación reducen las dimensiones de las características, conservando la información más relevante. Las capas completamente conectadas, al final de la red, realizan la clasificación basada en las características aprendidas.

B. Aplicaciones de las CNN

Las aplicaciones de las CNN abarcan desde el reconocimiento de objetos y rostros hasta la clasificación de imágenes médicas y la detección de anomalías en datos visuales. Su capacidad para aprender jerarquías de características hace que sean especialmente efectivas en tareas de visión por computadora.

C. Base de datos MNIST

La base de datos MNIST es un conjunto de datos que consiste en imágenes en escala de grises de dígitos escritos a mano, cada una etiquetada con el número correspondiente. La tarea común asociada con MNIST es la clasificación de dígitos.

D. One-Hot Encoding

One-Hot Encoding es una técnica que se utiliza para representar datos categóricos, como las etiquetas en la clasificación, de una manera que sea adecuada para su procesamiento por algoritmos de aprendizaje automático. Cada categoría se representa mediante un vector binario donde solo un elemento es 1, indicando la clase a la que pertenece.

E. Base de Datos CIFAR10

CIFAR10 es otro conjunto de datos utilizado para tareas de clasificación de imágenes. Contiene 60,000 imágenes en color distribuidas en 10 clases diferentes, como automóviles, aviones, gatos, perros, etc.

F. Overfitting

El overfitting ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos. Identificar y abordar el overfitting es crucial para lograr un modelo robusto.

G. Consideraciones finales

La elección de arquitecturas de red, optimizadores y estrategias de entrenamiento puede tener un gran impacto en el rendimiento de las CNN. Además, la exploración de diversas técnicas, como el aumento de épocas o el tamaño del batch, puede mejorar aún más la capacidad de generalización del modelo.

III. metodología/implementacion

Se comenzó con la implementación de una función normalizadora de imágenes que recibe números enteros [0,255] y entrega una salidas flotantes en el intervalo [0,1]. Luego de la normalización de lo anterior se desarrolla un código para ampliar la dimensión de la entrada con la función expanddims a cada matriz de tamaño (28,28) para dejarla como (28,28,1)

Luego para el procesamiento de las label se implementa una función one-hot encoding que se encargo de codificar un vector de números en una matriz de k clases. Lo que debemos tener en cuenta es que esta matriz de k clases cada fila representa una muestra del vector originales decir, la posición del 1 indica la clase a la que pertenece esa muestra.

Finalmente se inicia la construcción de la red neuronal artificial con convoluciones y max-pooling. Se inicia con la implementación de net1 que es una red con entrada de dimensión igual a sampleshape, dos capas internas de 32 y 64 kernel maps, tamaño de kernel 3, stride de 1, padding de 0 y activación relu. Para terminar con una capa para reducir los feature maps utilizando max-pooling, una de aplanamiento con flatten y una ultima de fully-connected para así conectar con las funciones prehechas del código, pudiendo retornar así 'model'.

Siguiendo con la parte 5 de la implementación del código, aquí se compila y entrena el modelo definido anteriormente, pero el objetivo al menos en esta sección solo era definir los hiperparametros 'batchsize' y 'epochs'. Se eligen 128 y 700 respectivamente. 128 por que cuadriduplica el numero predeterminado de 32, pero tarda mas al utilizar un mayor numero de ejemplos de entrenamiento en cada iteración. Tambien es elegido 700 por que es cercano al punto en el que los resultados comienzan a converger y a ser muy costosos en cuanto a tiempo.

De manera similar a las dos anteriores partes, en la parte 6 se define net2 que sera la segunda red neuronal, similar a net1 con la diferencia de que en este caso se remueve la capa de maxpooling y se añade un stride igual a 2 al segundo bloque. Luego con los mismos numeros elegidos para los hiperparametros anteriores, se compila y entrena una red para net2.

Para la segunda parte de la implementacion de esta tarea, se implemento una cnn para CIFAR10, apoyandose en los desarrollos anteriores, para esto se siguieron los suigientes 4 pasos:
1) Se codifica el one-hot utilizando la que se creo anteriormente para codificar y.train e y.test

2)Luego se normalizo la imagen en x.train y x.test

3)Se creo la CNN para CIFAR10 junto a su resumen, para net.1 con adam y adadelta

4) Finalmente y apoyado en lo anterior se entrena el modelo creado en CIFAR10, partiendo por entrenar la red con 30 epoch, con un batch.size de 128, un validation.split de 0.2, los mismos prints de las funciones anteriores brindadas en la estructura del codigo.

Para finalizar con la metodología detrás de la implementación del código que se acaba de describir, se analizan los resultados impresos y se sacan conclusiones teniendo en cuenta los conocimientos previos adquiridos en las clases.

IV. Resultados

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

 conv2d (Conv2D)             (None, 26, 26, 32)        320

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d (MaxPooling2  (None, 12, 12, 64)        0
 D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

(a)

Fig. 1: resumen de net.1 con batch.size=128 y epochs=700

```
Epoch 698/700
422/422 - 3s - loss: 0.0306 - accuracy: 0.9918 - val_loss: 0.0585 - val_accuracy: 0.9840 - 3s/epoch - 7ms/step
Epoch 699/700
422/422 - 3s - loss: 0.0305 - accuracy: 0.9919 - val_loss: 0.0582 - val_accuracy: 0.9837 - 3s/epoch - 7ms/step
Epoch 700/700
422/422 - 3s - loss: 0.0305 - accuracy: 0.9918 - val_loss: 0.0579 - val_accuracy: 0.9842 - 3s/epoch - 8ms/step
================================================================================
Assesing Test dataset...
================================================================================
Test loss: 0.051665373146533966
Test accuracy: 0.9829000234603882
```

(a)

Fig. 2: Resultados net.1 con batch.size=128 y epochs=700

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

# Construir una red
model = net_2(sample_shape, 10)
model.summary()
```

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

 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 conv2d_3 (Conv2D)           (None, 12, 12, 64)        18496

 flatten_1 (Flatten)         (None, 9216)              0

 dense_2 (Dense)             (None, 128)               1179776

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 1199882 (4.58 MB)
Trainable params: 1199882 (4.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

(a)

Fig. 3: Resumen net.2 con batch.size=128 y epochs=700

```
Epoch 698/700
422/422 - 2s - loss: 0.0350 - accuracy: 0.9906 - val_loss: 0.0646 - val_accuracy: 0.9810 - 2s/epoch - 4ms/step
Epoch 699/700
422/422 - 2s - loss: 0.0349 - accuracy: 0.9907 - val_loss: 0.0648 - val_accuracy: 0.9815 - 2s/epoch - 4ms/step
Epoch 700/700
422/422 - 2s - loss: 0.0349 - accuracy: 0.9907 - val_loss: 0.0648 - val_accuracy: 0.9810 - 2s/epoch - 4ms/step
================================================================================
Assesing Test dataset...
================================================================================
Test loss: 0.06904061883687973
Test accuracy: 0.9793999791145325
```

(a)

Fig. 4: Resultados net.2 con batch.size=128 y epochs=700

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

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

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

 conv2d_4 (Conv2D)           (None, 30, 30, 32)        896

 conv2d_5 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten_2 (Flatten)         (None, 12544)             0

 dense_4 (Dense)             (None, 128)               1605760

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 1626442 (6.20 MB)
Trainable params: 1626442 (6.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

(a)

Fig. 5: Resumen CIFAR10 con net..1 con batch.size=128 y epochs=30

```
Epoch 30/30
313/313 - 3s - loss: 0.0327 - accuracy: 0.9889 - val_loss: 2.8623 - val_accuracy: 0.6551 - 3s/epoch - 10ms/step
================================================================================
Evaluando el conjunto de prueba...
================================================================================
Pérdida en el conjunto de prueba: 3.0063834190368652
Precisión en el conjunto de prueba: 0.6468999981880188
```

(a)

Fig. 6: Resultados CIFAR10 con optimizador adam

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

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

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_6 (Conv2D)           (None, 30, 30, 32)        896

 conv2d_7 (Conv2D)           (None, 28, 28, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 flatten_3 (Flatten)         (None, 12544)             0

 dense_6 (Dense)             (None, 128)               1605760

 dense_7 (Dense)             (None, 10)                1290

=================================================================
Total params: 1626442 (6.20 MB)
Trainable params: 1626442 (6.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

(a)

Fig. 7: Resultados net..1 con batch.size=128 y epochs=30

```
Epoch 30/30
313/313 - 3s - loss: 1.9161 - accuracy: 0.3472 - val_loss: 1.9237 - val_accuracy: 0.3399 - 3s/epoch - 10ms/step
================================================================================
Evaluando el conjunto de prueba...
================================================================================
Pérdida en el conjunto de prueba: 1.9200888872146606
Precisión en el conjunto de prueba: 0.3393999934196472
```

(a)

Fig. 8: Resultados CIFAR10 con optimizador adadelta

A. Caso MNIST con net_1 (Batch Size: 128, Épocas: 700)

- Test Loss: 0.0516653
- Test Accuracy: 0.9829000

B. Caso MNIST con net_2 (Batch Size: 128, Épocas: 700, Sin Max-Pooling)

- Test Loss: 0.0690406
- Test Accuracy: 0.9793999

C. Caso CIFAR10 con net_1 (Batch Size: 128, Épocas: 30, Optimizador: Adam)

- Test Loss: 3.0063834
- Test Accuracy: 0.6468999

D. Caso CIFAR10 con net_1 (Batch Size: 128, Épocas: 30, Optimizador: Adadelta)

- Test Loss: 1.92008888
- Test Accuracy: 0.3393999

V. Análisis

A. ¿Qué modelo funciona mejor?

En el caso de MNIST, `net_1` funciona ligeramente mejor en términos de precisión y pérdida. En el caso de CIFAR10, `net_1` con el optimizador Adam tiene un rendimiento superior en comparación con `net_1` con Adadelta.

B. ¿Qué optimizador funciona mejor?

En los casos proporcionados, el optimizador Adam parece funcionar mejor para CIFAR10.

C. ¿Existe alguna evidencia de overfitting?

Como sabemos el overfitting se refiere a que un modelo estadístico está sobreajustado cuando lo entrenamos con muchos datos. Cuando un modelo se entrena con tantos datos, comienza a aprender del ruido y las entradas de datos inexactas en nuestro conjunto de datos

Entonces, para comprobar este caso se hizo correr el codigo esta vez en lugar de 700 epoch solo con 30 para comparar y ver si existe tal evidencia.

```
================================================================================
Assesing Test dataset...
================================================================================
Test loss: 0.28671833872795105
Test accuracy: 0.9178000092506409
```

Fig. 9: Resultados net.1 con batch.size=128 y epochs=30.

```
================================================================================
Assesing Test dataset...
================================================================================
Test loss: 0.3190581202507019
Test accuracy: 0.9117000102996826
```

Fig. 10: Resultados net.2 con batch.size=128 y epochs=30.

Como podemos observar la perdida y precisión da peores resultados para números bajos de epoch por lo que diremos que no existen indicios de que el modelo presente problemas al entrenarlo con tantos datos(700 en este caso)

D. ¿Cómo podemos mejorar aún más el rendimiento?

- Se podrían cambiar en los hiperparámetros, como el batch.size, los epoch y la arquitectura de la red(mas capas?).
- También se podrían probar otros optimizadores y funciones de pérdida.
- En el caso de CIFAR10, se podría considerar el uso de técnicas de aumento de datos para mejorar el rendimiento del modelo en un conjunto de datos relativamente pequeño.

VI. Conclusiones Generales

En esta tarea, se realizaron diversas implementaciones y experimentos con redes neuronales convolucionales (CNN) utilizando los conjuntos de datos MNIST y CIFAR-10. A continuación, se presentan las conclusiones generales:

- **Desempeño en MNIST:**
  - La red net.1 superó ligeramente a net.2 en términos de precisión y pérdida en MNIST.
  - El modelo con un tamaño de lote de 128 y 700 épocas alcanzó una precisión del 98.29
  - La omisión de capas de max-pooling en net.2 no proporcionó una mejora significativa en este escenario.
- **Desempeño en CIFAR-10:**
  - La elección del optimizador tuvo un impacto significativo, y `net_1` con Adam superó a la versión con Adadelta en términos de precisión y pérdida.
  - El modelo con Adam alcanzó una precisión del 64.69
- **Evaluación General:**
  - No hay evidencia clara de overfitting en los resultados presentados.
  - Los hiperparámetros son importantes para mejorar el rendimiento.
  - La normalización de imágenes y la codificación one-hot se aplicaron para preparar los datos de entrada.

En resumen, los experimentos proporcionan insights valiosos para la configuración de modelos CNN en tareas de clasificación, ofreciendo una base para futuras investigaciones y ajustes en el diseño de redes neuronales.

VII. Bibliografía

Ayudantía 5:
https://ucampus.uoh.cl/uoh/2023/2/COM4402/1/material_docente/detalle?id=2535477
Ayudantía 6:
https://ucampus.uoh.cl/uoh/2023/2/COM4402/1/material_docente/detalle?id=2545149
chat-gpt:
https://chat.openai.com