CONVOLUTIONAL NEURAL NETWORKS

Romel Vázquez
ITESM Campus Monterrey
Ingeniero de Software
Monterrey, Nuevo León, 64700
A01700519@tec.mx
Github: https://github.com/RomelVazquez2008/HomeworkAssignmentConvolutionalNeuralNetworks.git

ABSTRACT

Las redes neuronales convolucionales son bastante efectivas para el procesamiento de imágenes, es por esto que se realizaron cuatro experimentos variando las arquitecturas y/o los hiper parámetros en el lenguaje de programación Python en conjunto de la librería Tensorflow, para comparar los valores de la precisión y la pérdida; de esta forma se demuestra la importancia y los casos en los cuales se debe de usar este tipo de aprendizaje profundo.

Palabras Clave: CNN, capas, modelo, experimentos, arquitectura, hiper parámetros.

1. INTRODUCCIÓN

Las redes neuronales son algoritmos que forman parte del aprendizaje profundo. Dentro de estas se encuentran las redes neuronales convolucionales (CNN) las cuales usualmente se usan para tareas de clasificación y visión artificial, debido a que aprovechan el algebra lineal y la multiplicación de matrices, para la clasificación de imágenes, como el reconocimiento de objeto. [1]

Las CNN contiene tres principales capas, las cuales son: convolucional donde ocurre la mayor parte del cálculo, agrupamiento que reduce la dimensionalidad y totalmente conectada para la conexión directa a un nodo de la capa anterior. Con cada capa, la CNN aumenta su complejidad e identifica mayores proporciones de imagen. [1]

2. METODOLOGÍA

Se hace uso de la aplicación de código abierto de Jupyter Notebook para la interacción del usuario con el código, se utiliza el lenguaje de programación Python versión 3.11.3, así como las librerías de "Tensorflow", "matplotlib" y "numpy". Los entrenamientos de los experimentos se realizaron con el conjunto de datos CIFAR-10.

3. EXPERIMENTOS

A continuación, se muestra cada uno de los experimentos realizados.

3.1 EXPERIMENTO 1

El propósito de este experimento (Exp. 1) es crear una base el cual tratará de ser superado en los demás experimentos.

Hiper parámetros:

- 10 épocas
- Tamaño de lote de 32
- Tasa de aprendizaje $10x10^{-3}$

Arquitectura:

- Capa convolucional de 32 filtros, núcleo de $3x3$, activación ReLU
- Capa agrupación máxima de $2x2$
- Capa plana
- Capa densa de 128 unidades y activación ReLU
- Capa densa de salida con activación "softmax"

```
# Experimento 1 del modelo base
# Capa convucional de 32 filtro, con un kernel de 3x3 y activación ReLU
# Agrupación Máxima de 2x2
# Capa Aplanada
#  Densidad de la capa con 128 unidades y activación ReLU
# Capa de salida con activacion softmax


modelo1 = models.Sequential([
    layers.Conv2D(32,   (3,   3), activation= RELU, input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
```

```
    layers.Flatten(),
    layers.Dense(128, activation= RELU),
    layers.Dense(10, activation= SOFTMAX)
])

modelo1.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Entrenamiento del Modelo
# Nota: Sea paciente con el entrenamiento de esta linea de codigo
modelo1Entrenado = False

if modelo1Entrenado == False:
    entrenamiento1 = modelo1.fit(EntrenamientoDeImagenes, EntrenamientoDeEtiquetas, epochs=10,
                     validation_data=(PruebaDeImagenes, PruebaDeEtiquetas))
    modelo1Entrenado = True

else:
    print("Modelo 1 ya entrenado previamente, cambie la condicional inicial si lo quiere re-entrenar")

# Precision del Modelo
resultadosModelo1 = modelo1.evaluate(PruebaDeImagenes, PruebaDeEtiquetas, verbose=2)
precisionModelo1 = resultadosModelo1 [1] * 100
perdidaModelo1 = resultadosModelo1 [0]

print('Precision del Modelo 1: %.2f%%' % precisionModelo1)
print(f'Perdida del Modelo 1: {perdidaModelo1:.2f}')
```

Experimento 1: Modelo Base

3.2 EXPERIMENTO 2

Se cambiará la arquitectura de este experimento (Exp. 2) con el fin de mejorar la precisión de este código a diferencia del Experimento 1.

Hiper parámetros:

- 10 épocas
- Tamaño de lote de 32
- Tasa de aprendizaje $10x10^{-3}$

Arquitectura:

- Capa convolucional de 32 filtros, núcleo de $3x3$, activación ReLU
- Capa de agrupación máxima de $2x2$
- Capa de abandono de 25%
- Capa densa de 128 unidades y activación ReLU
- Capa adicional de convolucional de 128 filtros, núcleo de $3x3$, activación ReLU
- Capa adicional de Agrupación máxima de $2x2$
- Capa Plana
- Capa densa de 128 unidades y activación ReLU
- Capa adicional de abandono de 50%
- Capa densa de salida con activación “softmax”

```
# Define the model with additional convolutional layers
modelo2 = models.Sequential([
    layers.Conv2D(32, (3, 3), activation= RELU, input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),
    layers.Conv2D(128, (3, 3), activation= RELU), #Buscar lo que hace esto
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation= RELU ), #buscar que hace esto
    layers.Dropout(0.5),
    layers.Dense(10, activation= SOFTMAX)
])

modelo2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Entrenamiento del Modelo 2
# Nota: Sea paciente con el entrenamiento de esta linea de codigo

modelo2Entrenado = False

if modelo2Entrenado == False:
```

```
    entrenamiento2 = modelo2.fit(EntrenamientoDeImagenes, EntrenamientoDeEtiquetas, epochs=10,
                    validation_data=(PruebaDeImagenes, PruebaDeEtiquetas))
    modelo2Entrenado = True

else:
    print("Modelo 2 ya entrenado previamente, cambie la condicional inicial si lo quiere re-entrenar")

# Precision del Modelo 2
#Poner Accuracy, Perdida
#Matriz de confusión
#Tabla comparativa
resultadosModelo2 = modelo2.evaluate(PruebaDeImagenes, PruebaDeEtiquetas, verbose=2)
precisionModelo2 = resultadosModelo2[1] * 100
perdidaModelo2 = resultadosModelo2[0]

print('Precision del Modelo 2: %.2f%%' % precisionModelo2)
print(f'Perdida del Modelo 2: {perdidaModelo2:.2f}')
```

Experimento 2: Modelo con variación de arquitectura

3.3 EXPERIMENTO 3

Para este experimento (Exp. 3) se explorarán diferentes hiper parámetros, con el fin de descubrir el mejor conjunto de estos para tener la mayor precisión posible.

Hiper parámetros:

- 5, 10 o 15 épocas
- Tamaño de lote de 32, 64 o 128
- Tasa de aprendizaje de $10x10^{-2}$, $10x10^{-3}$ o $10x10^{-4}$

Arquitectura:

- Capa convolucional de 32 filtros, núcleo de $3x3$, activación ReLU
- Capa agrupación máxima de $2x2$
- Capa plana
- Capa densa de 128 unidades y activación ReLU
- Capa densa de salida con activación “softmax”

```
#Definicion de los hyperparamteros
aprendizajes = [0.01, 0.001, 0.0001]
lotes = [32, 64, 128]
epocas = [5, 10, 15]

entrenamientos = []
#Entrenamiento de los modelos con distintos hyper paramteros
#Se usan la misma arquitectura del modelo 1, ya que esta es la más rapida de entrenar
modelo3Entrenado = False

if modelo3Entrenado == False:
    for lote in lotes:
        for epoca in epocas:
            for aprendizaje in aprendizajes:
                modelo3 = models.Sequential([
                    layers.Conv2D(32, (3, 3), activation= RELU, input_shape=(32, 32, 3)),
                    layers.MaxPooling2D((2, 2)),
                    layers.Flatten(),
                    layers.Dense(128, activation= RELU),
                    layers.Dense(10, activation= SOFTMAX)
                ])
                optimizador = tf.keras.optimizers.Adam(learning_rate=aprendizaje)
                modelo3.compile(optimizer=optimizador,loss='sparse_categorical_crossentropy',metrics=['accuracy'])

                entrenamiento3 = modelo3.fit(EntrenamientoDeImagenes, EntrenamientoDeEtiquetas, epochs=epoca, batch_size=lote, validation_split=0.2, verbose=0)
                resultadosModelo3 = modelo3.evaluate(PruebaDeImagenes, PruebaDeEtiquetas, verbose=0)
                precisionModelo3 = resultadosModelo3 [1]
                perdidaModelo3 = resultadosModelo3 [0]
                entrenamientos.append((aprendizaje, lote, epoca, precisionModelo3, perdidaModelo3))
    modelo3Entrenado = True

else:
    print("Modelo 3 ya entrenado previamente, cambie la condicional inicial si lo quiere re-entrenar")
```

```
34.
35. for entrenamiento in entrenamientos:
36.     print("Experimento con
    hiperparametros: tasa de aprendizaje
    ->", entrenamiento[0], " tamanio del
    lote ->", entrenamiento[1], "
    cantidad de epocas ->",
    entrenamiento[2]," precision de ->
    %", round(entrenamiento[3]*100,2), "
    perdida de ->",
    round(entrenamiento[4],2))
37. #Despliegue del mejor modelo con los
    mejores hyper parametros
38. mejorResultado = max(entrenamientos,
    key=lambda x: x[3])
39. print('Los mejores hiperparametros
    del modelo 3 tienen una tasa de
    aprendizaje de: ',
    mejorResultado[0], ', tamanio de
    lote de: ', mejorResultado[1], ',
    cantidad de epocas: ',
    mejorResultado[2], ', precision de:
    %', round(mejorResultado[3] *100,2),
    ',perdida de: ',
    round(mejorResultado[4],2) )
```

Experimento 3: Modelo con variación de hiper parámetros

3.4 EXPERIMENTO 4

En este último experimento (Exp. 4) se implementaron dos técnicas de normalización avanzadas, las cuales son: normalización de Batch y aumento de datos.

Hiper parámetros:

- 10 épocas
- Tamaño de lote de 64
- Tasa de aprendizaje $10x10^{-3}$
- Aumento de Datos
    - Rango de rotaciones de 15
    - Rango de desplazamiento horizontal de $10x10^{-1}$
    - Rango de desplazamiento vertical de $10x10^{-1}$
    - Inversión horizontal de imágenes

Arquitectura:

- Capa convolucional de 32 filtros, núcleo de $3x3$, activación ReLU
- Capa de normalización de Batch
- Capa agrupación máxima de $2x2$
- Capa plana
- Capa densa de 128 unidades y activación ReLU
- Capa de normalización de Batch
- Capa densa de salida con activación “softmax”

```
1.  #Aplicación de la normalización
    de Batch
2.  #Se aplica a la arquitectura del
    Modelo 1 porque es el más
    sencillo de todos los Modelos
3.  modelo4 = models.Sequential([
4.      layers.Conv2D(32, (3, 3),
    activation= RELU,
    input_shape=(32, 32, 3)),
5.      layers.BatchNormalization(),
6.      layers.MaxPooling2D((2, 2)),
7.      layers.Flatten(),
8.      layers.Dense(128, activation=
    RELU),
9.      layers.BatchNormalization(),
10.     layers.Dense(10, activation=
    SOFTMAX)
11. ])
12.
13. modelo4.compile(optimizer='adam',
14.               loss='sparse_catego
    rical_crossentropy',
15.               metrics=['accuracy'
    ])
16. #Definición del aumento de datos
17. aumentoDeDatos =
    ImageDataGenerator(
18.     rotation_range=15,
19.     width_shift_range=0.1,
20.     height_shift_range=0.1,
21.     horizontal_flip=True,
22. )
23. aumentoDeDatos.fit(EntrenamientoD
    eImagenes)
24. #Entrenamiento del Modelo 4
25. modelo4Entrenado = False
26.
27. if modelo4Entrenado == False:
28.     entrenamiento4 =
    modelo4.fit(aumentoDeDatos.flow(E
    ntrenamientoDeImagenes,
    EntrenamientoDeEtiquetas,
    batch_size=64), epochs=10,
29.                       validation_da
    ta=(PruebaDeImagenes,
    PruebaDeEtiquetas))
30.
31. else:
32.     print("Modelo 4 ya entrenado
    previamente, cambie la
    condicional inicial si lo quiere
    re-entrenar")
33. # Precision del Modelo 4
34. resultadosModelo4 =
    modelo4.evaluate(PruebaDeImagenes
    , PruebaDeEtiquetas, verbose=2)
35.     modelo4Entrenado = True
```

```
precisionModelo4 =
    resultadosModelo4 [1] * 100
perdidaModelo4 = resultadosModelo4
    [0]

print('Precision del Modelo 4:
    %.2f%%' % precisionModelo4)
print('Perdida del Modelo 2: ',
    round(perdidaModelo4,2))
```

Experimento 4: Modelo con normalización de técnicas avanzadas

4. RESULTADOS

Con las pruebas realizadas se obtuvieron los siguientes resultados (Tabla 1), en los cuales se puede reflejar precisión y pérdida de validación de datos.

Experimento #	Precisión	Perdida
1	63.82%	1.13
2	69.81%	0.87
3 (aprendizaje: $10x10^{-2}$, lote: 32, épocas: 5)	10%	2.31
3 (aprendizaje: $10x10^{-3}$, lote: 32, épocas: 5)	60.68%	1.13
3 (aprendizaje: $10x10^{-4}$, lote: 32, épocas: 5)	55.73%	1.26
3 (aprendizaje: $10x10^{-2}$, lote: 32, épocas: 10)	10%	2.3
3 (aprendizaje: $10x10^{-3}$, lote: 32, épocas: 10)	60.76%	1.28
3 (aprendizaje: $10x10^{-4}$, lote: 32, épocas: 10)	60.44%	1.13
3 (aprendizaje: $10x10^{-2}$, lote: 32, épocas: 15)	35.75%	1.82
3 (aprendizaje: $10x10^{-3}$, lote: 32, épocas: 15)	62.08%	1.42
3 (aprendizaje: $10x10^{-4}$, lote: 32, épocas: 15)	62.57%	1.07
3 (aprendizaje: $10x10^{-2}$, lote: 64, épocas: 5)	36.51%	1.8
3 (aprendizaje: $10x10^{-3}$, lote: 64, épocas: 5)	59.24%	1.17
3 (aprendizaje: $10x10^{-4}$, lote: 64, épocas: 5)	54.87%	1.29
3 (aprendizaje: $10x10^{-2}$, lote: 64, épocas: 10)	38.56%	1.74
3 (aprendizaje: $10x10^{-3}$, lote: 64, épocas: 10)	62.72%	1.15
3 (aprendizaje: $10x10^{-4}$, lote: 64, épocas: 10)	58.24%	1.2
3 (aprendizaje: $10x10^{-2}$, lote: 64, épocas: 15)	46.85%	2.14
3 (aprendizaje: $10x10^{-3}$, lote: 64, épocas: 15)	64.01%	1.3
3 (aprendizaje: $10x10^{-4}$, lote: 64, épocas: 15)	61%	1.11
3 (aprendizaje: $10x10^{-2}$, lote: 128, épocas: 5)	49.63%	1.45
3 (aprendizaje: $10x10^{-3}$, lote: 128, épocas: 5)	60.27%	1.15
3 (aprendizaje: $10x10^{-4}$, lote: 128, épocas: 5)	50.31%	1.43
3 (aprendizaje: $10x10^{-2}$, lote: 128, épocas: 10)	54.24%	1.82
3 (aprendizaje: $10x10^{-3}$, lote: 128, épocas: 10)	64.03%	1.06
3 (aprendizaje: $10x10^{-4}$, lote: 128, épocas: 10)	57.53%	1.23

3 (aprendizaje: $10x10^{-2}$, lote: 128, épocas: 15)	39.3%	1.71
3 (aprendizaje: $10x10^{-3}$, lote: 128, épocas: 15)	62.23%	1.13
3 (aprendizaje: $10x10^{-4}$, lote: 128, épocas: 15)	60.01%	1.14
4	70.26%	0.86

Tabla 1: Resultados de los experimentos

5. DISCUSIÓN

Del experimento 3, se considerará como válido el que tiene los hiper parámetros de: aprendizaje de $10x10^{-3}$, lote de 128 y épocas de 10; con precisión de 64.03% y pérdida de 1.06.

Con los datos anteriores se puede notar que el modelo con la menor precisión de los cuatro experimentos fue el experimento 1 con una precisión de 63.82% ya que este era el modelo base a superar, mientras que el modelo con una mejor precisión fue el experimento 4 con una precisión 70.26% que tuvo mejoras considerables en los hiper parámetros y la arquitectura; las técnicas avanzadas de normalización de Batch y aumento de datos.

Lo mismo aplica para el valor de la perdida dado que el experimento 1 tiene la mayor de todas, siendo esta de 1.13. Los experimentos 2 y 4 tuvieron los valores más bajos de perdida, siendo de: 0.87 y 0.86 respectivamente.

Se puede dar a conocer que las mejoras en las arquitecturas de los modelos mejoran en mayor medida la precisión y la perdida, a diferencia de cambiar los hiper parámetros

6. CONCLUSIÓN

La variación de los hiper parámetros y la arquitectura dependerá del científico de datos, así como de los componentes del sistema con el que se esté entrenando un modelo. [2]

Si se tiene un hiper parámetro muy alto (como la tasa de aprendizaje) este modelo podría hacer que el modelo aprenda muy rápido con resultados que no sean correctos, o si es un valor muy bajo el modelo tardaría demasiado en dar un resultado. [2]

En el caso de la arquitectura con una mayor cantidad de dimensiones en las capas se puede tener una mayor precisión, aunque el modelo se puede ver expuesto a un caso sobreajuste, por esto hay que reconocer el uso de abandono de capas para evitar dichos casos. [3]

7.REFERENCIAS

[1] *¿Qué son las redes neuronales convolucionales?* | IBM. (s. f.). https://www.ibm.com/mxes/topics/convolutional-neural-networks

[2] *¿Qué es el ajuste de hiperparámetros? - Explicación de los métodos de ajuste de hiperparámetros* - AWS. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/hyperparameter-tuning/#:~:text=optimizaci%C3%B3n%20de%20hiperpar%C3%A1metros.-,%C2%BFPor%20qu%C3%A9%20es%20importante%20el%20ajuste%20de%20hiperpar%C3%A1metros%3F,modelo%20para%20lograr%20resultados%20%C3%B3ptimos

[3] *Rocha, X.* (2022, 1 diciembre). Clasificación de imágenes con redes neuronales convolucionales. Medium. https://medium.com/@a01706707/clasificaci%C3%B3n-de-im%C3%A1genes-con-redes-profundas-a6a980b9cea5

[4] *Cruz-Duarte, J. M.* (s. f.). Deep learning. Canvas Tec de Monterrey. https://experiencia21.tec.mx/courses/463404/files/178003100?module_item_id=28471420