Universidad de O'Higgins

Tarea N° 3:
Redes neuronales convolucionales

José Tomás Garciá Risco

COM4402 – Introducción a Inteligencia Artificial

Escuela de Ingeniería, Universidad de O'Higgins

11, Septiembre, 2023

***Abstract*—Esperamos aplicar redes neuronales convolucionales que logre clasificar los labels correspondientes a la base de datos de manuscritos MNIST y la base de datos de imágenes pequeñas CIFAR-10**

Palabras Clave: Redes neuronales convolucionales,MNIST ,CIFAR-10 ,Modelos, One-Hot

I. Introduction

Las Redes Neuronales Convolucionales (CNN) han revolucionado el campo de la visión por computadora y el procesamiento de imágenes, mostrando un rendimiento excepcional en tareas de clasificación y detección de patrones en datos visuales. Estas redes, inspiradas en la organización visual del cerebro, son especialmente eficaces en la extracción de características jerárquicas de datos complejos, como imágenes. A lo largo del tiempo, diversas arquitecturas de CNN han emergido, cada una diseñada para abordar desafíos específicos en la clasificación de imágenes.

En nuestra exploración, utilizaremos dos modelos de CNN, denominados net_1 y net_2, que representan enfoques distintos en la construcción de arquitecturas. net_1 incorpora capas de convolución seguidas de capas de max-pooling, lo que permite la extracción eficiente de características y la reducción de dimensionalidad. Por otro lado, net_2, al eliminar las capas de pooling y ajustar los strides, se enfoca en preservar más información espacial en el proceso de convolución.

Ambos modelos finalizan con capas fully-connected que realizan la clasificación. Estas redes se caracterizan por su capacidad para aprender automáticamente características discriminativas de las imágenes durante el entrenamiento, permitiendo una representación eficiente de patrones complejos en los datos.

En cuanto al preprocesamiento, normalizamos las imágenes y aplicaremos codificación one-hot a las etiquetas para asegurar que la red pueda aprender y generalizar eficazmente en la tarea de clasificación de imágenes.

En la aplicación práctica de estos modelos en el conjunto de datos CIFAR-10, nuestro enfoque será evaluar cómo las diferentes arquitecturas afectan el rendimiento de clasificación. Durante el entrenamiento, monitoreamos las métricas de pérdida y precisión para entender la capacidad de generalización de cada modelo.

II. Marco Teorico

- Red neuronal: Las redes neuronales son un modelo computacional evolucionado a partir de diversas contribuciones científicas. Consiste en un conjunto de unidades llamadas neuronas artificiales, conectadas entre sí para transmitir señales.
- Redes Neuronales Convolucional: son un tipo de red neuronal diseñada específicamente para procesar datos bidimensionales, como imágenes. Se destacan por su capacidad para aprender automáticamente jerarquías de características a través de capas de convolución y pooling.
- Capas de Convolución: Aplican filtros a pequeñas regiones de la entrada para detectar patrones locales.

- Capas de Pooling: Reducen la dimensionalidad de los feature maps, manteniendo la información más relevante.
- Capas Fully-Connected: Realizan la clasificación final basada en las características extraídas.
- One-hot:Es una técnica utilizada para representar etiquetas categóricas,cada clase se representa con un vector de longitud igual al número de clases, con un 1 en la posición correspondiente a la clase y 0 en las demás posiciones.

III. METODOLOGÍA

Empezamos corriendo parte del código base importando las librerías a usar preparando el ambiente a usar para empezar la parte 1 la creación de una función de normalización de imágenes.

```
def normalize_images(images):
    images = images.astype(float)
    images /= 255.0

    return images

### *No* modificar las siguientes líneas
test_normalize_images(normalize_images)

# Normalizar los datos para su uso futuro
x_train = normalize_images(x_train)
x_test = normalize_images(x_test)
```

```
Pruebas superadas.
Puede pasar a la siguiente tarea.
```

Lo que hace es tomar una matriz de imágenes y realiza dos pasos para prepararlas para su uso en tareas de aprendizaje automático: convierte los valores a tipo float y normaliza los valores dividiéndolos por 255.0.

Luego para la parte 2 Ampliar la dimensión de la entrada usamos expand_dims de numpy para para agregar una nueva dimensión a las matrices. formando las dimensiones pedidas

- La dimensionalidad de `x_train` debe ser $(60000, 28, 28, 1)$
- La dimensionalidad de `x_test` debe ser $(10000, 28, 28, 1)$

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

```
Shape of x_train (60000, 28, 28, 1)
Shape of y_train (60000,)
Shape of x_test (10000, 28, 28, 1)
Shape of y_test (10000,)
```

Para el paso 3 codificamos one hot encoding técnica utilizada para representar etiquetas categóricas

```
def one_hot(vector, num_classes):
    num_samples = len(vector)
    one_hot = np.zeros((num_samples, num_classes))

    for i in range(num_samples):
        one_hot[i, vector[i]] = 1

    return np.array(one_hot)


### *No* modifique las siguientes lineas ###
test_one_hot(one_hot)

# One-hot codifica los labels de MNIST
y_train = one_hot (y_train, 10)
y_test = one_hot(y_test, 10)
```

```
Pruebas superadas.
Puede pasar a la siguiente tarea.
```

Pasamos al paso 4 donde implementamos una red convolucional con las características pedidas para crear el primer modelo net1 que presenta 2 2 capas de convolución la primera tiene 32 filtros, cada uno de tamaño 3x3, utiliza la función de activación ReLU la segunda es similar pero contiene 64 en vez de 32 después se aplica una capa de max-pooling con un tamaño de ventana de 2x2 pasamos a aplanar el mapa de características resultante después del max-pooling para prepararlo a la capa fully connected con 128 neuronas y activación ReLU, junto a la capa de salida con un número de

neuronas igual al número de clases y terminamos definiendo el modelo.

```
def net_1(sample_shape, nb_classes):
    input_x = Input(shape=sample_shape)
    conv1 = Conv2D(32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='rel
    conv2 = Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='rel
    pool = MaxPooling2D(pool_size=(2, 2), padding='same')(conv2)
    flatten = Flatten()(pool)
    fc1 = Dense(128, activation='relu')(flatten)
    x = Dense(nb_classes, activation='softmax')(fc1)
    model = Model(inputs=input_x, outputs=x)
    return model
```

entramos al paso 5 definir los Hiper Parámetros y entrenar la red en este caso decidí usar un batch de 64 y 20 epocas esperando no crear overfitting con un loss aproximado a 0.3 y un val loss 0.24 asumí que eran resultados aceptables.

```
Epoch 20/20
844/844 - 5s - loss: 0.2989 - accuracy: 0.9133 - val_loss: 0.2344 - val_accuracy: 0.93
```

Paso 6 creamos un segundo modelo net2 el cual se asemeja bastante al primero pero este presenta un stride mayour en la segunda capa de convolución lo que tiende a reducir la dimensionalidad de la salida además de eliminar el max pooling lo que permite que la red conserve características locales más finas en los feature maps. Esto puede ser beneficioso en tareas donde la información espacial detallada es crucial.

Paso 7 definimos los Hiper Parámetros de este modelo opte a usar los mismos para poder comparar resultados en un ambiente similar

```
Epoch 20/20
844/844 - 4s - loss: 0.3027 - accuracy: 0.9126 - val_loss: 0.2366 - val_accuracy: 0.93
==========================================================================
```

encontramos parámetros muy parecidos pasamos al paso 8 aplicando lo usado ahora en la base de datos CIFAR10 el cual consta de 60000 imágenes en color de 32×32 , compuesta por 10 clases, con 6000 imágenes por clase aplicando one hot en los labels

```
num_classes = 10
y_train = one_hot(y_train, num_classes)
y_test = one_hot(y_test, num_classes)

### *No* modifique las siguientes líneas ###
# Imprima los tamaños de los datos (variables)
print('Shape of x_train {}'.format(x_train.shape))
print('Shape of y_train {}'.format(y_train.shape))
print('Shape of x_test {}'.format(x_train.shape))
print('Shape of y_test {}'.format(y_train.shape))
```

```
Shape of x_train (50000, 32, 32, 3)
Shape of y_train (50000, 10)
Shape of x_test (50000, 32, 32, 3)
Shape of y_test (50000, 10)
```

Paso 9 normalizamos las imágenes.

```
x_train = normalize_images(x_train)
x_test = normalize_images(x_test)
```

Paso 10 creamos la CNN aunque los resultados de net1 y net2 son prácticamente similares net1 muestra una ligera mejora parte del tiempo en algunos parámetros aunque sea por poco deje el modelo Adam.

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

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

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

Paso 11 entrenamos el modelo y definimos Hiper Parámetros

```
# Construya el código dentro de esta celda
epochs = 30
batch_size = 128

# Entrenar el modelo
logs = model.fit(x_train, y_train,
                 epochs=epochs,
                 batch_size=batch_size,
                 validation_split=0.2,
                 )
```

```
- loss: 0.0399 - accuracy: 0.9865 - val_loss: 3.3047 - val_accuracy: 0.620
-------------------------
```

IV. RESULTADOS

- net1: presentó alta accuracy y val_accuracy superando el 0.9 en ambos casos el aumento de epochs bajo un poco más el loss y el val loss los cuales están en los alrededores del 0.3 sino un poco más bajos pero en niveles mínimos de 0.02 en algunos de los mejores casos antes de terminar en overfit
- net2: presentó resultados prácticamente similares con alta accuracy y val_accuracy superando el 0.9 en ambos además de loss y el val loss los alrededor del 0.3
- CIFAR10: Muy buena accuracy y muy baja loss pero presenta gran val_loss la cual va subiendo entre épocas lo que puede significar overfit y la val_accuracy deja que desear

V. ANÁLISIS

- ¿Qué modelo funciona mejor?: net1 presenta resultados ligeramente mejores lo que puede significar que el dataset no se beneficia de la reducción de dimensionalidad.
- ¿Qué optimizador funciona mejor?: En este caso Adam esto puede ser por que la situación no presenta las características que benefician a los otros estas ciendo datos escasos para Adagrad y problemas no estacionarios para Adadelta.
- ¿Existe alguna evidencia de overfitting?: En CIFAR10 se puede deducir la existencia de overfitting ya que val_loss se vuelve mayor con el paso de las épocas acompañado de una reducción de val_accuracy
- ¿Cómo podemos mejorar aún más el rendimiento? Hay múltiples métodos como Regularización , ajuste de Hyper Parámetros y cambio de arquitectura entre otros.

VI. CONCLUSIONES

Al final pudimos comparar la efectividad de modelos y optimizadores con efectividad en algunos casos mas que otros más notablemente en los optimizadores.