Redes de Neuronas Convolucionales en Pytorch

Diego Andrade Canosa

Roberto López Castro

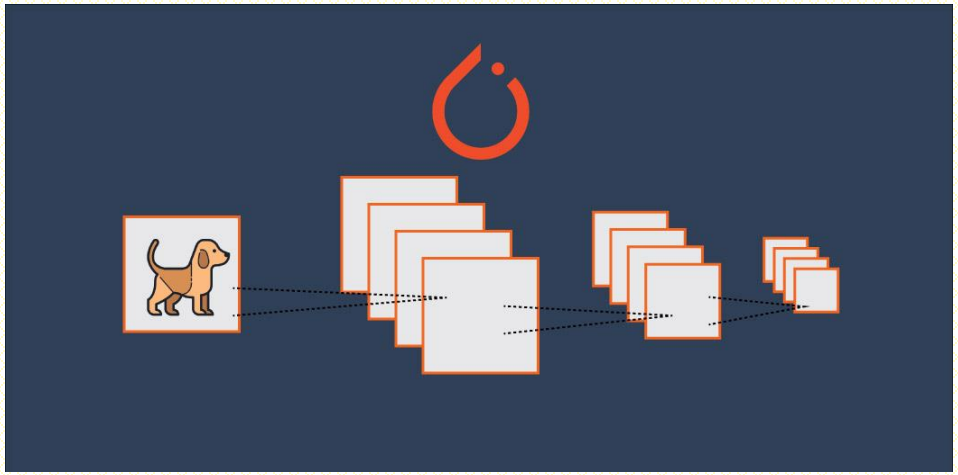
Índice

- Introducción
- Conceptos básicos de CNNs
- CNNs en Pytorch

Introducción

- Las redes neuronales convolucionales (CNNs) han revolucionado el campo del aprendizaje profundo y la visión por computadora (e.g., reconocimiento de imágenes, detección de objetos y segmentación semántica)

- En esta presentación, exploraremos los conceptos básicos de las CNNs y cómo se pueden implementar en PyTorch



citic

¿Qué son?

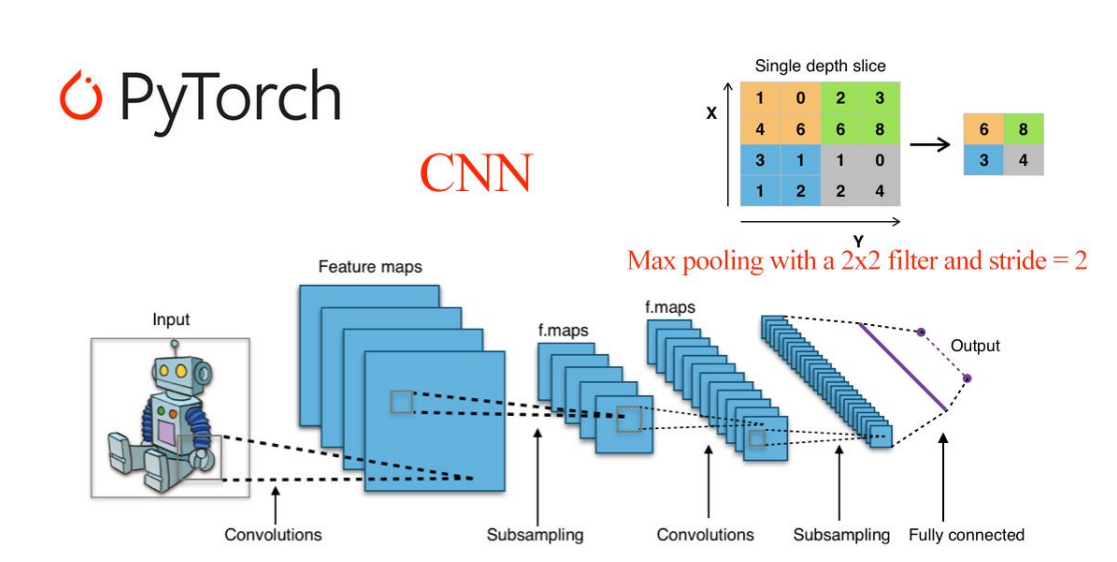
• Las redes neuronales convolucionales (CNNs) son un tipo de arquitectura de redes neuronales artificiales inspiradas en el funcionamiento del sistema visual humano.

•Están especialmente diseñadas para el procesamiento eficiente de datos en forma de rejillas estructuradas, como imágenes o señales de audio.

•Las CNNs se basan en el concepto de convolución, que es una operación matemática fundamental para extraer características relevantes de los datos.

¿Qué son?

- A diferencia de las redes neuronales tradicionales, las CNNs utilizan capas convolucionales y capas de pooling para aprender y representar características locales de manera jerárquica.
- Las características aprendidas por las capas convolucionales se combinan en capas totalmente conectadas para realizar tareas de clasificación, detección de objetos, segmentación semántica, entre otras.



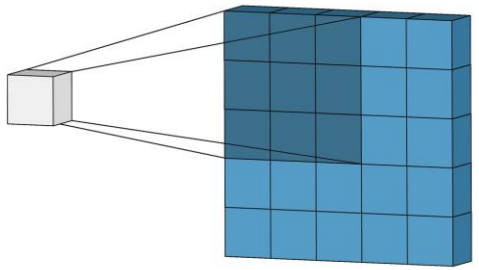
citic

Arquitectura básica de una CNN

- Capas de entrada: Reciben los datos de entrada, generalmente imágenes.
- Capas convolucionales: Extraen características locales mediante filtros convolucionales.
- Capas de activación: Introducen no linealidad en la red mediante funciones como ReLU.
- Capas de pooling: Reducen la dimensionalidad de las características preservando las más relevantes.
- Capas totalmente conectadas: Combinan características para tareas específicas.
- Capa de salida: Produce la salida final de la red (clasificación, detección, etc.).

Convolución

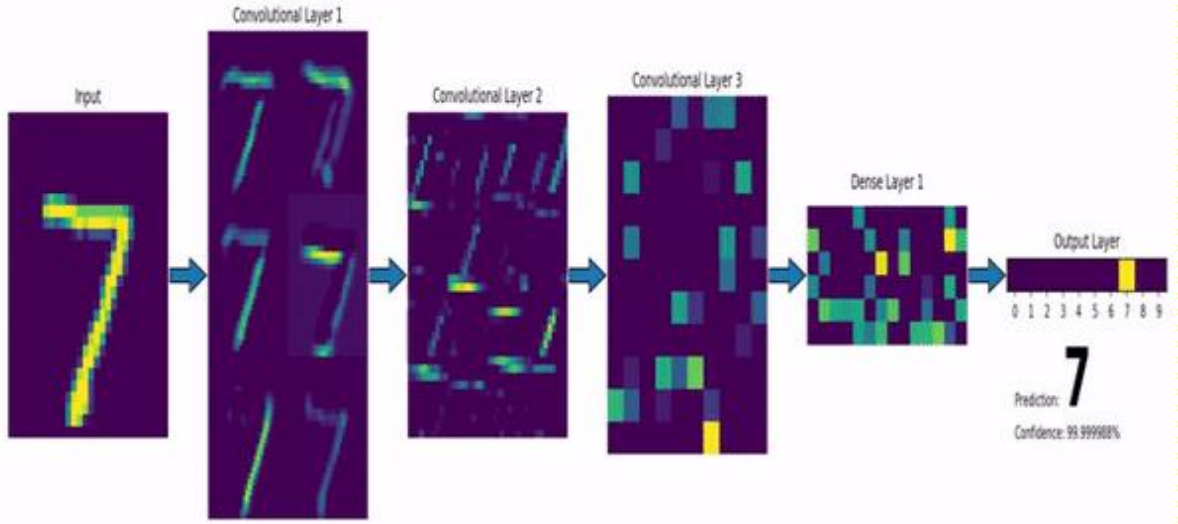
- La convolución es una operación fundamental en las redes neuronales convolucionales (CNNs).
- Permite extraer características locales al aplicar un filtro convolucional a una región de la entrada.
- El filtro se desliza sobre la entrada aplicando una multiplicación de elementos y sumando los resultados.
- La convolución es capaz de capturar patrones y características relevantes, como bordes, texturas y formas.



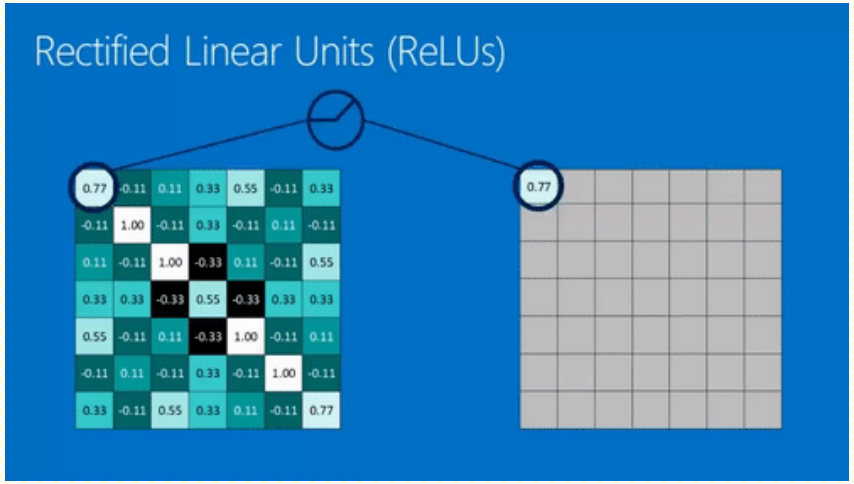
citic

Convolución

- Los filtros convolucionales se aprenden durante el entrenamiento de la CNN y se aplican en paralelo a múltiples ubicaciones de la entrada.
- La convolución se utiliza en las capas convolucionales de una CNN para generar mapas de características que representan las características aprendidas.



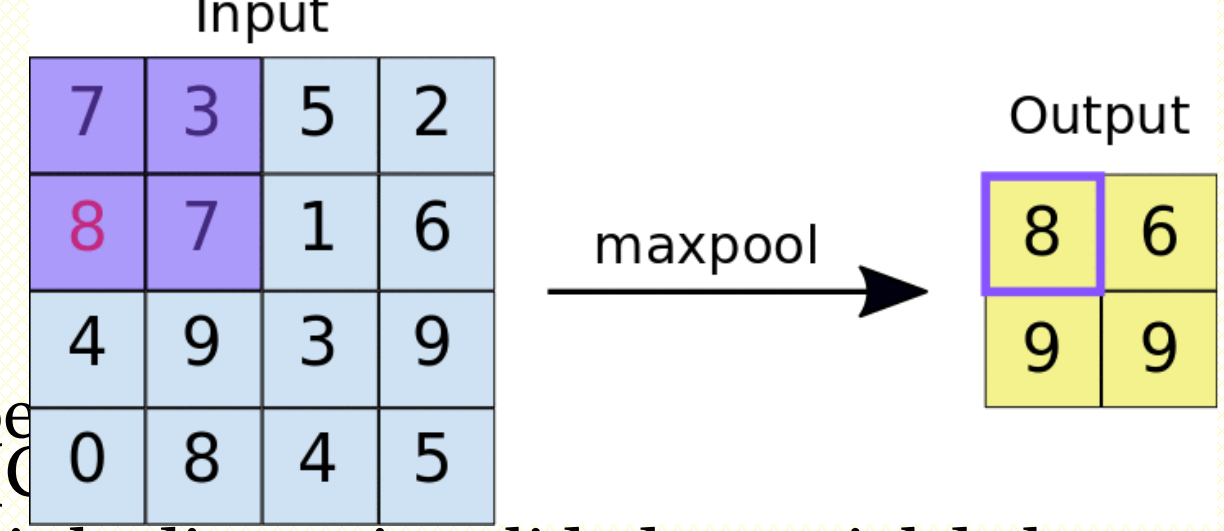
Función de activación



- Las funciones de activación son elementos clave convolucionales (CNNs).
- Introducen no linealidad en la red, permitiendo modelar relaciones y patrones más complejos.
- Una función de activación comúnmente utilizada en las capas convolucionales de una CNN es la función ReLU (Rectified Linear Unit).
- La función ReLU mantiene los valores positivos sin cambios y establece a cero los valores negativos.
- Esto permite una activación más rápida y eficiente, promoviendo la convergencia durante el entrenamiento.
- Otras funciones de activación populares incluyen la función sigmoide y la tangente hiperbólica, aunque son menos comunes en las capas convolucionales de las CNNs.

citic

Pooling



- Las capas de pooling desempe
neuronales convolucionales (C
- Su función principal es reducir la dimensionalidad espacial de las características.
- El pooling se realiza mediante operaciones como el máximo o el promedio en una región local de la entrada.
- Al reducir la resolución espacial, las capas de pooling ayudan a disminuir la cantidad de parámetros y a generar invariantes a pequeñas traslaciones o deformaciones.
- Esto permite una extracción de características más robusta y eficiente.
- Sin embargo, es importante equilibrar la reducción de dimensionalidad con la preservación de información relevante.

Implementación de CNNs en PyTorch

- PyTorch es ampliamente utilizado para implementar CNNs debido a su flexibilidad y facilidad de uso.
- Proporciona una API intuitiva para construir y entrenar modelos de CNN.
- Los pasos básicos para implementar una CNN en PyTorch son:
  - Definir la estructura de la red.
  - Definir la función de pérdida.
  - Configurar el optimizador.
  - Entrenar el modelo.
  - Evaluar el modelo.
- PyTorch ofrece herramientas y utilidades adicionales para facilitar el desarrollo de modelos de CNN.

Conjuntos de datos populares para CNNs

- Los conjuntos de datos populares son fundamentales en el entrenamiento de redes neuronales convolucionales (CNNs).
- Algunos conjuntos de datos comunes incluyen:
  - MNIST: Dígitos escritos a mano para clasificación.
  - CIFAR-10 y CIFAR-100: Imágenes de objetos en 10 y 100 clases respectivamente.
  - ImageNet: Millones de imágenes etiquetadas en diversas categorías.
  - COCO: Detección y segmentación de objetos en imágenes.
  - Pascal VOC: Detección, clasificación y segmentación de objetos.
- Estos conjuntos de datos son accesibles y útiles para proyectos de CNNs.

Ejemplo de una CNN básica

- Capa de entrada: Imágenes en escala de grises de tamaño 28x28 píxeles.
- Capa convolucional: 32 filtros de 3x3, con función de activación ReLU.
- Capa de pooling: Reduce la dimensionalidad a la mitad.
- Capa convolucional: 64 filtros de 3x3, con función de activación ReLU.
- Capa de pooling: Reduce la dimensionalidad a la mitad.
- Capa completamente conectada: 128 unidades, con función de activación ReLU.
- Capa de salida: 10 unidades para la clasificación de 10 clases (en el caso de MNIST).

- Durante el entrenamiento, se utiliza la función de pérdida de entropía cruzada y el optimizador de descenso de gradiente estocástico (SGD)

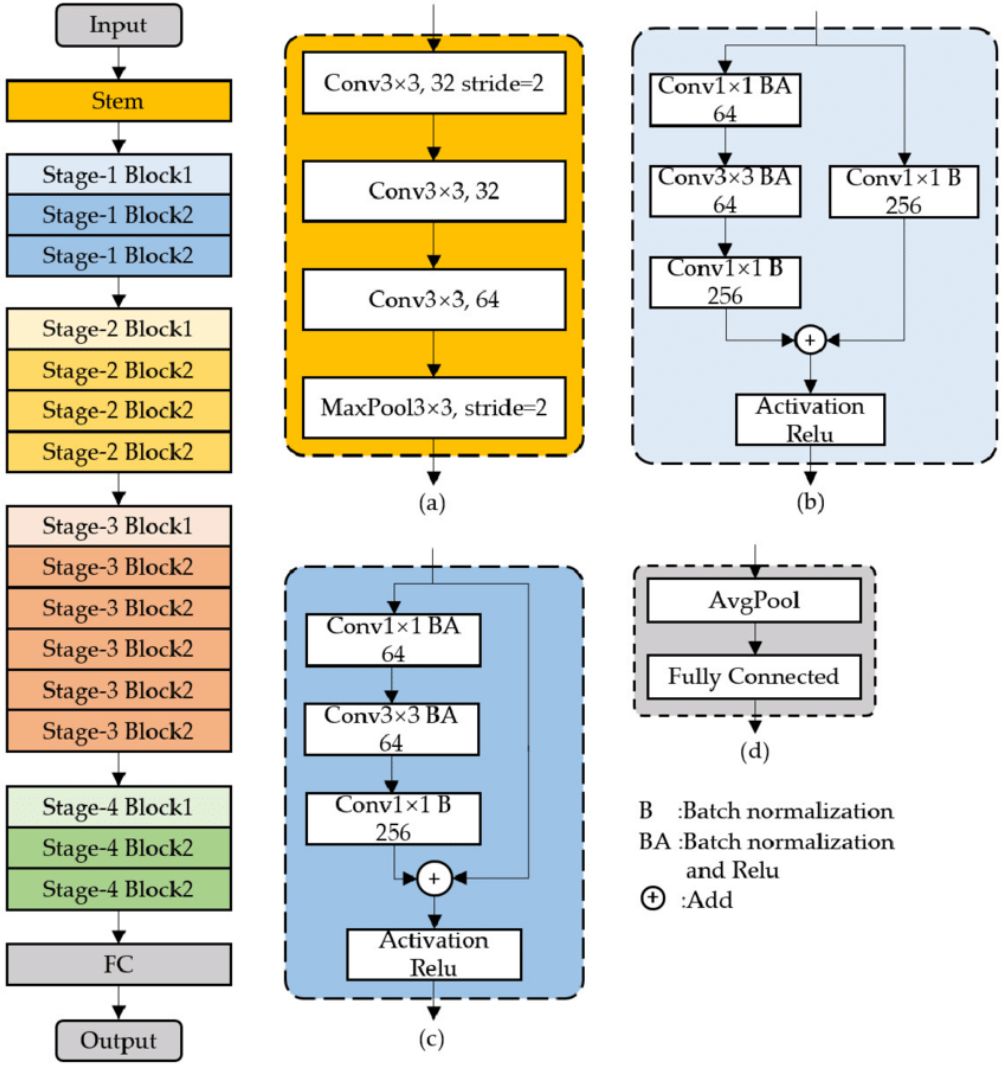
```python
import torch
import torch.nn as nn

# Definición de la estructura de la red
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3)  # Capa convolucional
        self.relu = nn.ReLU()  # Función de activación ReLU
        self.pool = nn.MaxPool2d(2, 2)  # Capa de pooling
        self.conv2 = nn.Conv2d(32, 64, 3)  # Capa convolucional
        self.fc1 = nn.Linear(64 * 6 * 6, 128)  # Capa completamente conectada
        self.fc2 = nn.Linear(128, 10)  # Capa de salida

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.pool(x)
        x = x.view(-1, 64 * 6 * 6)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Creación de una instancia del modelo
model = CNN()
```

Arquitecturas avanzadas de CNN

- Arquitecturas avanzadas de CNN han demostrado un gran rendimiento en tareas de visión por computadora.
- Algunas arquitecturas populares son:
  - AlexNet
  - VGGNet
  - ResNet
  - InceptionNet
  - EfficientNet
- Estas arquitecturas se han utilizado con éxito en clasificación, detección y segmentación de imágenes.

- Actualmente los Transformers están obteniendo buenos resultados en este ámbito (ViT) - próximamente

Arquitecturas avanzadas de CNN. ResNet-50



Funciones de pérdida comunes

- Pérdida de regresión de media cuadrática (nn.MSELoss): Utilizada en regresión. Calcula la pérdida como el promedio de las diferencias al cuadrado entre las predicciones y los valores reales.

- Pérdida de divergencia de Kullback-Leibler (nn.KLDivLoss): Utilizada para medir discrepancias entre distribuciones de probabilidad. Aplicable en generación de imágenes o aprendizaje no supervisado.

Funciones de pérdida comunes

• Pérdida de entropía cruzada (nn.CrossEntropyLoss): Utilizada en clasificación multiclase. Calcula la pérdida basada en la diferencia entre las probabilidades predichas y las etiquetas verdaderas.

•Pérdida de entropía cruzada binaria (nn.BCELoss): Utilizada en clasificación binaria. Calcula la pérdida según la diferencia entre las predicciones y las etiquetas verdaderas.

Optimización

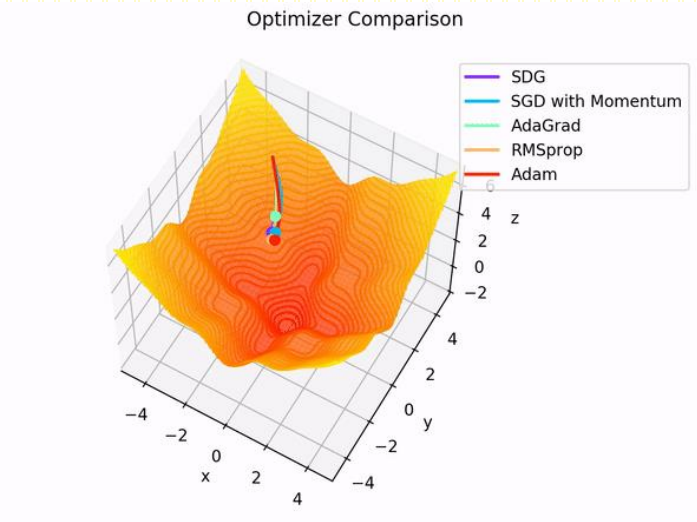
•SGD (Stochastic Gradient Descent): Algoritmo básico de optimización que actualiza los parámetros en la dirección opuesta al gradiente, utilizando un subconjunto aleatorio de los datos en cada iteración.

•Adam (Adaptive Moment Estimation): Algoritmo de optimización adaptativo que ajusta la tasa de aprendizaje de forma individual para cada parámetro. Utiliza estimaciones del primer y segundo momento de los gradientes para calcular las actualizaciones de los parámetros.

Optimización

•RMSprop (Root Mean Square Propagation): Algoritmo de optimización que adapta la tasa de aprendizaje de forma individual para cada parámetro. Realiza actualizaciones basadas en promedios móviles de los gradientes cuadrados anteriores.

•AdaGrad (Adaptive Gradient): Algoritmo de optimización que adapta la tasa de aprendizaje de forma individual para cada parámetro. Realiza actualizaciones escalando el gradiente de cada parámetro según la suma acumulativa de los gradientes anteriores.



Optimización

- import torch.optim as optim
-
- # Ejemplo de optimización con SGD
- optimizer_sgd = optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)
-
- # Ejemplo de optimización con Adam
- optimizer_adam = optim.Adam(model.parameters(), lr=learning_rate)
-
- # Ejemplo de optimización con RMSprop
- optimizer_rmsprop = optim.RMSprop(model.parameters(), lr=learning_rate)
-
- # Ejemplo de optimización con AdaGrad
- optimizer_adagrad = optim.Adagrad(model.parameters(), lr=learning_rate)

Entrenamiento de una CNN en PyTorch

- Lab 1: 03_train_cnn.ipynb

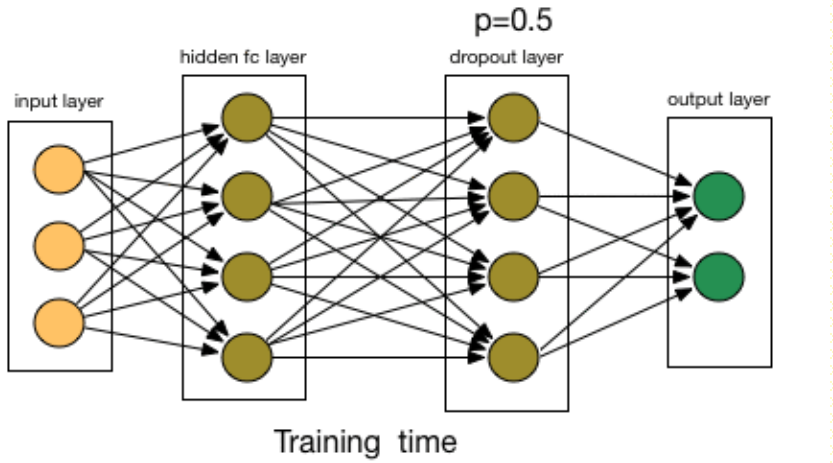
Regularización

•La regularización es una técnica utilizada en el entrenamiento de redes neuronales convolucionales para evitar el sobreajuste y mejorar la generalización del modelo. A continuación, se presentan dos técnicas de regularización comunes en PyTorch:

•Regularización L2:
  •También conocida como regularización de peso decaído o "weight decay".
  •Consiste en agregar un término de penalización a la función de pérdida que penaliza los valores grandes de los pesos del modelo.
  •Ayuda a controlar la complejidad del modelo y evitar la sobreoptimización.

Regularización

- Dropout:
  - Es una técnica de regularización que introduce aleatoriamente la desactivación temporal de algunas unidades (neuronas) durante el entrenamiento.
  - Al desactivar unidades aleatoriamente, se evita la dependencia excesiva entre unidades y se promueve la generalización.
  - PyTorch proporciona el módulo nn.Dropout para aplicar dropout en capas específicas del modelo.



citic

Regularización

```
import torch.nn as nn
import torch.optim as optim

# Regularización L2 (Weight Decay)
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=0.001)

# Dropout
dropout_rate = 0.5
model = nn.Sequential(
    nn.Linear(input_size, hidden_size),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.Linear(hidden_size, output_size)
)
```

Evaluación de una CNN en PyTorch

- Exactitud (Accuracy): Se puede calcular utilizando la función torch.eq() para comparar las etiquetas predichas con las etiquetas verdaderas, y luego calcular el promedio de aciertos.

```
correct = torch.eq(predictions, labels).sum().item()
accuracy = correct / total_samples
```

- Matriz de confusión (Confusion Matrix): Se puede utilizar la función sklearn.metrics.confusion_matrix() para obtener la matriz de confusión, que muestra los resultados de clasificación por clase.

```
from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(labels, predictions)
```

Evaluación de una CNN en PyTorch

- Precisión, sensibilidad y puntuación F1: Se pueden calcular utilizando la función sklearn.metrics.classification_report(), que proporciona un resumen detallado de las métricas por clase.

```
from sklearn.metrics import classification_report
classification_report = classification_report(labels, predictions)
```

- Curva ROC y área bajo la curva (AUC): Para problemas de clasificación binaria, se puede calcular la curva ROC y el AUC utilizando la función sklearn.metrics.roc_curve() y sklearn.metrics.auc() respectivamente.

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(labels, scores)
roc_auc = auc(fpr, tpr)
```

Visualización de características en PyTorch

La visualización de características es una técnica útil para comprender y analizar el comportamiento interno de una red neuronal convolucional (CNN). PyTorch proporciona varias formas de visualizar las características aprendidas por una CNN. A continuación, se presentan algunas técnicas comunes:

- Mapas de activación:
  - Los mapas de activación muestran las áreas del mapa de características que se activan para una determinada entrada.
  - Se pueden visualizar activaciones en diferentes capas para comprender cómo evoluciona la información a medida que se propaga a través de la red.
  - Se puede acceder a los mapas de activación utilizando ganchos (hooks) en PyTorch

Visualización de características en PyTorch

La visualización de características es una técnica útil para comprender y analizar el comportamiento interno de una red neuronal convolucional (CNN). PyTorch proporciona varias formas de visualizar las características aprendidas por una CNN. A continuación, se presentan algunas técnicas comunes:

- Filtros visuales:
  - Los filtros visuales representan los pesos aprendidos por las capas convolucionales.
  - Visualizar los filtros puede ayudar a comprender qué características específicas está buscando la CNN en una imagen.
  - Los filtros se pueden obtener accediendo a los parámetros de las capas convolucionales.

Visualización de características en PyTorch

- Representaciones de características:
  - Las representaciones de características se refieren a la proyección de las características de alta dimensión en un espacio de menor dimensión.
  - Técnicas como el análisis de componentes principales (PCA) o t-SNE se pueden utilizar para visualizar las representaciones de características.

- Activaciones de neuronas específicas:
  - Se pueden seleccionar neuronas específicas dentro de una capa y visualizar su respuesta a diferentes entradas.
  - Esto puede ayudar a identificar patrones o características particulares que una neurona en particular está aprendiendo.

Visualización de características en PyTorch

Lab 2: 03_visualization.ipynb