CI-0148 Aprendizaje Máquina
Prof. Pablo Sauma Chacón

Laboratorio 5

Objetivo: Familiarizar al estudiante con la creación y el uso de redes neuronales utilizando la biblioteca Torch. Así como la construcción de redes neuronales sofisticadas, como es el caso de una red convolucional para clasificación de imágenes.

Enunciado: Lea las instrucciones del documento y resuelva el enunciado en Python. Al inicio de su documento adjunte su nombre y carnet como un comentario. Este trabajo es de carácter individual, tampoco se aceptarán códigos que no hayan sido desarrollados por su persona.

Para este laboratorio se trabajará con el set de datos CIFAR-10, que contiene 50000 imágenes de entrenamiento y 10000 de prueba de tamaño 32x32 (x3 dado el RGB) con imágenes pertenecientes a 10 categorías: avión, automóvil, pájaro, gato, perro, venado, rana, caballo, barco y camión. Su objetivo es crear un sistema que logre clasificar correctamente al menos el 60% de las imágenes del conjunto de prueba.

Para ello deberá implementar la clase **`ConvNN`** que corresponde a una red neuronal utilizando los módulos de Torch. Puede usar la arquitectura que usted desee pero se le brindan los siguientes consejos/propuestas de pasos:

1. El profesor le provee el conjunto de datos y un código de carga que le permitirá cargar los datos de entrenamiento y prueba a la memoria, junto al arreglo de nombres para cada una de las clases. Además posee un método para graficar imágenes del conjunto de datos y poder ver su contenido (una vez que haya ejecutado el paso 2).

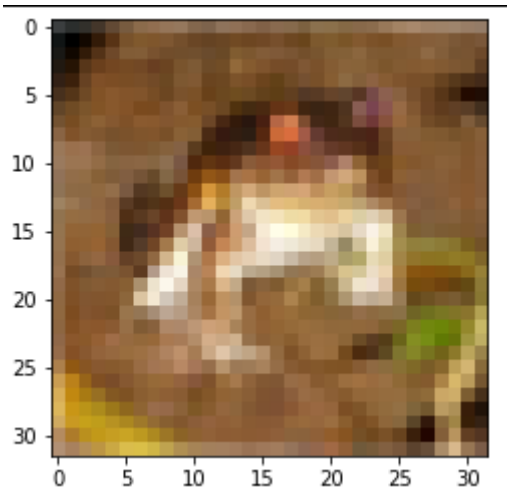

Imagen de entrenamiento 0, categoría: rana

2. Note que los datos de las imágenes vienen como un arreglo de datos (tipo uint8) de tamaño 50000x3096, donde cada fila corresponde a una imagen y los 3096 valores

corresponden a una imagen de 3x32x32 (los primeros 1024 valores corresponden a la imagen 32x32 del canal rojo, los siguientes 1024 al 32x32 del verde, y por último los 1024 del canal azul). Los labels corresponden a un arreglo de enteros tamaño 50000. En ambos casos deberá convertir los datos a tensores, y en el caso de las imágenes cambiar su forma de 50000x3096 a 50000x3x32x32 (note que el orden de los datos permanece igual, esto porque las capas convolucionales prefieren que los canales vengan como dimensión superior: CxHxW en lugar del habitual HxWxC; esto por motivos de eficiencia).

3. Deberá construir su red neuronal usando la biblioteca Torch según lo visto en clase. Dado que trabajaremos con imágenes **no** es recomendable usar capas densas hasta haber disminuido la complejidad del problema utilizando capas convolucionales. Las capas convolucionales en torch se crean de la siguiente manera:

```
nn.Conv2d(in_channels, out_channels, kernel_size)
```

`in_channels` corresponde a los canales de entrada (para la primera capa esto sería 3, dado que la imagen posee los 3 canales RGB), `out_channels` es la cantidad de capas de salida. Se suele aumentar la cantidad de capas en cada iteración, dado que cada capa buscará un "patrón" diferente en la imagen. El `kernel_size` se refiere al tamaño que observa dicha convolución, un kernel de 3x3 o de 5x5 son buenas alternativas para iniciar. Además puede jugar con el `stride`/desplazamiento de la convolución (por defecto es 1). Por último el `padding` representa el "borde" que se le agrega a la imagen (de manera que, al aplicar la convolución, la imagen resultante sea de las mismas dimensiones HxW que la original). Si el `stride` está en 1, se puede utilizar `padding=¨same¨` para que se elija el padding automáticamente de manera que el HxW de salida sea igual al original.

Una arquitectura habitual al procesar imágenes es utilizar una capa convolucional que aumenta la cantidad de canales, aplicar la función de activación y luego aplicar algún tipo de pooling como maxpool. Esto reducirá el tamaño de la imagen en las dimensiones HxW, reduciendo la complejidad del problema a pesar del incremento en profundidad. En torch el pooling se puede aplicar como:

```
nn.MaxPool2d(2)
```

Que haría un pooling por cada sección de 2x2 de la "imagen".

A manera de ejemplo si se aplica una `Conv2d(3, 5, (3,3))` activado con `LRelu` y luego un `MaxPool2d(2)`, cada una de nuestras imágenes pasaría de tener dimensiones 3x32x32 (3096) a tener dimensiones 5x16x16 (1280), y la cantidad de pesos que habría que ajustar para la capa convolucional sería de solo: 5x(3x3x3 + 1) = (140).

Por este motivo es recomendable aplicar múltiples capas de convolución (siempre aumentando la profundidad para extraer más "características") hasta llevar al

problema a una dimensión manejable para capas densas/completamente conectadas. Una vez que se llega a este nivel, será necesario “aplanar” los datos, por ejemplo, si nuestra imagen final es de tamaño 4x4x15 (240), será necesario convertirla en un tensor de 240 para poder enviarlo a la capa densa. Dado que estamos realizando clasificación de 10 categorías, se recomienda que su capa de salida sea de 10 neuronas (*one-hot encoding*) y utilizar activación softmax para el resultado final.

Por último, recuerde que la biblioteca Torch está altamente optimizada para manejar tensores, por lo que resultaría eficiente enviar todas las imágenes de un solo (NxCxHxW) en lugar de enviarlas una por una. Pero en ese caso, al aplanar el tensor para pasar los datos a la capa densa, deberá convertirlo en un tensor de tamaño NxX en lugar de solo tamaño X.

4. Otra recomendación es que utilice una función de pérdida apropiada para clasificación, como lo es `nn.CrossEntropyLoss()`. Esta pérdida recibe un tensor de tamaño NxC y otro de tamaño N (con valor enteros [0,C[) y calcula la pérdida adecuadamente.

5. Por último, puede utilizar cualquier optimizador que considere le puede ser útil.

6. Si su algoritmo dura mucho en cada época puede ser que no está reduciendo la dimensionalidad lo suficiente. Intente precalcular el tamaño/dimensión de cada una de sus capas para estimar adecuadamente la complejidad del problema.

7. Por último, cuando tenga su clase lista, entrénela por 100-250 épocas, o hasta que logre clasificar más del 60% de las imágenes del conjunto de prueba (no debe usar estos datos para entrenar la red, solo para medir su rendimiento). Guarde su modelo con el estado de su red, optimizador y época actual. Si su entrenamiento tomara mucho tiempo puede ejecutarlo por partes, cargando el número de época actual para llevar el control (pero no debería tomarle más de 15-60 minutos entrenar todo el modelo (*aplican restricciones)). Para pruebas iniciales puede usar subconjuntos de los datos de entrenamiento, como solo 5000 imágenes, para que sea más rápido y fácil ver el comportamiento de su estructura.

8. Si se encuentra satisfecho con su red, entonces grafique la matriz de confusión para las diferentes categorías. ¿Cuáles categorías confunde su red? ¿Por qué cree que esas categorías le generan confusión/errores de clasificación?