Tarea Deep Learning. Clasificación de imágenes.

Contents

1 Clasifciación de imágenes con una red profunda y Keras/TF

Para esta tarea, utilizaremos el conjunto de datos 0 y otro a elegir de entre: (recomendable 1 y 2 por nuestras capacidades de cálculo)

1. **Fashion MNIST**, que es un poco más complejo que el MNIST original de dígitos, proporcionando un desafío interesante para la clasificación de imágenes. Fashion MNIST consta de 70,000 imágenes en escala de grises de 28x28 píxeles, divididas en 10 categorías de ropa. Similar a la tarea anterior, esta actividad tiene como objetivo explorar diferentes técnicas y configuraciones en el entrenamiento de redes neuronales profundas.

2. **CIFAR-100**: Similar al CIFAR-10 pero con 100 clases en lugar de 10. Cada clase contiene 600 imágenes, para un total de 60,000 imágenes de 32x32 píxeles. El aumento en el número de clases introduce una complejidad adicional en comparación con CIFAR-10 y Fashion MNIST.

3. **SVHN (Street View House Numbers)**: Este conjunto de datos proviene de imágenes de números de casas recogidas por Google Street View. Contiene más de 600,000 dígitos en contexto real, lo que lo hace más complicado que los dígitos manuscritos de MNIST debido a la variabilidad de fondos, iluminación y orientación.

4. **Imagenet**: Uno de los conjuntos de datos más conocidos y desafiantes, contiene más de 14 millones de imágenes que han sido clasificadas en más de 20,000 categorías. Debido a su tamaño y complejidad, entrenar modelos en ImageNet es una tarea significativa que a menudo requiere mucha potencia computacional y tiempo.
5. **MS COCO (Microsoft Common Objects in Context)**: Es un conjunto de datos de detección de objetos, segmentación y subtítulos de imágenes que contiene más de 200,000 imágenes etiquetadas con 80 categorías de objetos. Es especialmente conocido por su complejidad debido a la presencia de múltiples objetos por imagen y a la variabilidad en el tamaño, la forma y la ubicación de los objetos.
6. **CelebA (Large-scale Celebrity Faces Attributes)**: Es un conjunto de datos de reconocimiento facial con más de 200,000 imágenes de celebridades, cada una con 40 anotaciones de atributos. La diversidad de las poses, las expresiones faciales y los accesorios hacen de este conjunto de datos un reto para los modelos de reconocimiento facial.

Estos conjuntos de datos presentan una variedad de desafíos que van desde la clasificación de imágenes hasta la detección de objetos y el reconocimiento facial, lo que los hace adecuados para proyectos más avanzados en visión por computadora y aprendizaje profundo. Cada uno de estos conjuntos de datos puede ayudar a los practicantes a desarrollar, probar y mejorar algoritmos de aprendizaje automático y técnicas de procesamiento de imágenes para abordar problemas del mundo real más complejos.

1.1 Objetivos de la Tarea

1. **Profundizar en el entendimiento del entrenamiento de redes neuronales en un conjunto de datos de clasificación de imágenes.**
2. **Experimentar con técnicas avanzadas de optimización, normalización y regularización.**
3. **Evaluar el impacto de diferentes configuraciones de red en el rendimiento del modelo.**

1.2 Descripción de la Tarea

Parte A: Construcción y Entrenamiento Inicial

1. **Construye una red neuronal profunda (DNN)** con las siguientes especificaciones:
   - **20 capas ocultas** con **100 neuronas** cada una.
   - Utiliza la **inicialización de Xavier/Glorot** y la **función de activación ELU**.
2. **Entrena la red** en el **conjunto de datos elegido** usando:
   - **Optimización Adam** y **detención temprana**.
   - Capa de salida **softmax** con **X(10,100,...) neuronas**.

Parte B: Integración de Mejoras y Comparación

1. **Implementa normalización por lotes** en tu red y **analiza las diferencias** en las curvas de aprendizaje con respecto a la configuración inicial. Considera:

- ¿Hay una mejora en la velocidad de convergencia?
- ¿El modelo resultante es superior?
- ¿Cuál es el impacto en el tiempo de entrenamiento?

2. **Sustituye la normalización por lotes con la activación ReLU** y ajusta la red para utilizar **inicialización He**. Compara el rendimiento y discute los resultados.

Parte C: Avanzando en Regularización y Optimización

1. **Introduce dropout** y evalúa cómo afecta al rendimiento del modelo. Luego, **aplica MC dropout** en la evaluación del modelo sin reentrenarlo. Analiza las diferencias.

```
#MCDropout, cómo aplicarlo en Keras
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(100, activation="relu",
                          kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(100, activation="relu",
                          kernel_initializer="he_normal"),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(10, activation="softmax")
])

# extra code - compile and train the model
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid))

# Crear un clase que hereda de Dropout

class MCDropout(tf.keras.layers.Dropout):
    def call(self, inputs, training=None):
        return super().call(inputs, training=True)

# extra code - shows how to convert Dropout to MCDropout in a Sequential model
# crear un modelo que copia las capas de otro, conviertiendo las que son DROPOUT
# A MCDropout
Dropout = tf.keras.layers.Dropout
mc_model = tf.keras.Sequential([
    MCDropout(layer.rate) if isinstance(layer, Dropout) else layer
    for layer in model.layers
```

```
])
mc_model.set_weights(model.get_weights())

mc_model.summary()

# usar MCDropout
# extra code - shows that the model works without retraining
tf.random.set_seed(42)
np.mean([mc_model.predict(X_test[:1])
         for sample in range(100)], axis=0).round(2)
```

1. **Experimenta con el optimizador SGD con momentum**. Reentrena tu modelo y observa si hay cambios notables en la precisión y la velocidad de entrenamiento en comparación con Adam.

1.3 Entregables

- **Código:** Deberás entregar el código fuente utilizado para construir, entrenar y evaluar tu modelo.
- **Informe:** Un documento que incluya:
  - Descripción de la arquitectura de la red y las configuraciones experimentadas.
  - Gráficos de las curvas de entrenamiento y validación para cada experimento.
  - Un análisis comparativo del rendimiento observado bajo diferentes configuraciones.
  - Reflexiones sobre cómo cada cambio afectó al modelo y por qué piensas que ocurrió esto.
  - Conclusiones y sugerencias para futuras exploraciones en este conjunto de datos.

1.4 Criterios de Evaluación

- **Organización y claridad del código.**
- **Implementación adecuada de las técnicas y configuraciones solicitadas.**
- **Calidad del análisis y profundidad de las reflexiones en el informe.**
- **Originalidad en las soluciones propuestas y en las mejoras experimentadas.**

Esta tarea está diseñada no solo para reforzar el conocimiento técnico en el entrenamiento de redes neuronales, sino también para fomentar el pensamiento crítico y analítico sobre cómo y por qué ciertas técnicas afectan el rendimiento del modelo. Se anima a los estudiantes a ir más allá de la tarea, explorando otras configuraciones, técnicas o incluso modificando la arquitectura de la red para ver cómo estos cambios pueden influir en los resultados finales.

1.5 Apéndice 1: Cargar datasets

Cargar conjuntos de datos como CIFAR-100, SVHN, MS COCO, o CelebA puede variar en complejidad debido a las diferencias en el formato de almacenamiento y la disponibilidad directa a través de APIs como la de TensorFlow/Keras. A continuación, te muestro cómo cargar cada uno de estos conjuntos de datos con ejemplos de código:

1.6 CIFAR-100

CIFAR-100 está directamente disponible en Keras, por lo que cargarlo es tan sencillo como:

```
from tensorflow.keras.datasets import cifar100

(x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='fine')
```

Aquí, `label_mode`'fine'= indica que estamos interesados en las 100 clases. Si quieres agruparlas en superclases, puedes cambiarlo a `label_mode`'coarse'=.

1.7 SVHN

SVHN no está disponible directamente a través de Keras, pero puedes usar TensorFlow Datasets (tfds), una biblioteca que facilita cargar y preprocesar conjuntos de datos en TensorFlow:

```
import tensorflow_datasets as tfds

svhn_train, svhn_test = tfds.load('svhn_cropped', split=['train', 'test'], as_sup
```

`'svhn_cropped'` indica que estás interesado en la versión recortada del conjunto de datos donde las imágenes ya están centradas en el dígito.

1.8 MS COCO

MS COCO requiere más trabajo para cargar, ya que no está directamente disponible a través de Keras o TensorFlow Datasets en un formato estándar. Sin embargo, puedes acceder a él a través de su API oficial o descargar el conjunto de datos desde la página web de COCO y luego cargarlo manualmente. Un ejemplo de cómo empezar con la API de COCO sería:

```
!pip install pycocotools
from pycocotools.coco import COCO

# Inicializar el objeto COCO para instancias de anotación
coco = COCO("path/to/annotations/instances_train2017.json")

# Ahora puedes usar el objeto coco para obtener ids de imágenes y cargar imágenes
```

1.9 CelebA

CelebA también está disponible a través de TensorFlow Datasets, lo que facilita su carga:

```
import tensorflow_datasets as tfds

celeba_train, celeba_test = tfds.load('celeb_a', split=['train', 'test'], as_supe
```

Recuerda que, especialmente para MS COCO y CelebA, cargar y procesar las imágenes y anotaciones puede requerir un código adicional para adaptar los datos a tu flujo de trabajo específico. Estos conjuntos de datos son significativamente más grandes y complejos que CIFAR-100 o SVHN, por lo que también debes tener en cuenta el rendimiento y la capacidad de memoria al trabajar con ellos.

###Cargar el conjunto de datos ImageNet puede ser más desafiante que los conjuntos de datos mencionados anteriormente debido a su considerable tamaño y porque no está disponible directamente a través de TensorFlow Datasets (tfds) o Keras en un formato fácil de usar. Sin embargo, hay formas de trabajar con ImageNet:

1.10 Descargar ImageNet

Primero, debes descargar el conjunto de datos de ImageNet. ImageNet es muy grande; por ejemplo, el conjunto de datos de ImageNet ILSVRC 2012 (el más utilizado) tiene aproximadamente 1.2 millones de imágenes de entrenamiento, 50,000 imágenes de validación y 100,000 imágenes de prueba, con 1,000 clases diferentes. La descarga requiere registrarse y obtener acceso a través del sitio web oficial de ImageNet: http://www.image-net.org/

1.11 Preparar el Conjunto de Datos

Una vez descargado, tendrás que descomprimir los archivos y organizarlos de manera que tu código pueda acceder a ellos de manera eficiente. Una estructura común es tener una carpeta para las imágenes de entrenamiento organizadas por clase y otra carpeta para las imágenes de validación con un archivo que mapee imágenes a sus clases.

1.12 Apéndice 2: Cargar ImageNet

Para cargar el conjunto de datos ImageNet, puedes usar la API `tf.data` de TensorFlow para crear un canal de entrada eficiente que pueda leer las imágenes desde el disco, aplicar preprocesamientos y aumentos de datos, y servirlas a tu modelo. Aquí te muestro un ejemplo básico de cómo podrías empezar a hacerlo:

```
import tensorflow as tf

# Suponiendo que tienes las imágenes en directorios por clase para entrenamiento
train_dir = "path/to/imagenet/train/"
validation_dir = "path/to/imagenet/validation/"

# Crear un dataset de TensorFlow a partir de las imágenes
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    label_mode='categorical',  # 'categorical' si tus etiquetas son one-hot encod
```

```
    image_size=(224, 224),  # Tamaño estándar de entrada para modelos preentrenad
    batch_size=32,
)

validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    validation_dir,
    label_mode='categorical',
    image_size=(224, 224),
    batch_size=32,
)

# Aplicar aumentos de datos o cualquier preprocesamiento necesario aquí
def preprocess(images, labels):
    # Aquí irían tus funciones de preprocesamiento
    return images, labels

train_dataset = train_dataset.map(preprocess)
validation_dataset = validation_dataset.map(preprocess)
```

1.13 Notas Finales

- Debido al tamaño de ImageNet, asegúrate de tener suficiente espacio en disco y memoria RAM.
- El preprocesamiento y el aumento de datos son pasos cruciales para trabajar con ImageNet, dado su tamaño y diversidad.
- Muchas veces, para experimentación rápida, investigadores y desarrolladores optan por utilizar versiones más pequeñas de ImageNet, como ImageNet-1K (que es parte de ILSVRC 2012), o conjuntos de datos preprocesados y más manejables disponibles en plataformas específicas de competencias como Kaggle.
- Algunos frameworks y bibliotecas pueden ofrecer scripts o herramientas para facilitar el trabajo con ImageNet, debido a su uso generalizado en la comunidad de aprendizaje profundo.