UNIVERSIDAD ADOLFO IBÁÑEZ

Tarea 3-TICS579

Deep Learning 2024-2

Profesor: Alfonso Tobar-Arancibia **Ayudante**: María Alejandra Bravo
Fecha de Entrega: 01/12/24 - 23:59 hrs. **Puntos Totales**: 36

INSTRUCCIONES:

- Entregue la tarea en una carpeta comprimida con el siguiente formato: **Tarea_1_AT_MB**. Donde en este caso, **AT** y **MB** corresponden a las iniciales de cada integrante del grupo. Se deben incluir todos los archivos entregados más los generados por usted.
- El notebook debe entregarse ejecutado con las celdas **en orden** y **sin errores de ejecución**.
- Recuerde que el código debe ser defendido en una sesión de defensa. **NO COPIE Y PEGUE CÓDIGO** que no entiende.
- No cumplir con las instrucciones implica nota 1.0.

Parte 0: Investigación

- Lea el siguiente **paper** correspondiente a la implementación de la Resnet. **Hint**: No es necesario que lea el paper completo, lea **Abstract**, **Introducción** y la Sección 3: **Deep Residual Learning** para entender la lógica detrás de la implementación de la Resnet.
- Lea el **quickstart** de la librería Gradio. Esta librería le permitirá crear un Demo para su modelo de Deep Learning.

Ojo: **Esta parte no tiene entregable asociado.**

Parte 1: Recreando la Resnet50 en Pytorch

Modifique **sólo el cuerpo** de cada una de las clases `Bottleneck()` y `Resnet()` en el archivo **resnet.py**. Utilice los nombres indicados para cada componente.

1. Complete la clase `Bottleneck()`. Consideramos un Bottleneck los 4 bloques conv2_x, conv3_x, conv4_x, conv5_x de la Tabla 1 del paper. Consideramos que cada `Bottleneck()` tiene 3 capas convolucionales de acuerdo al paper original. La 3era capa toma un `expanded_channels` que es 4 veces el `out_channels` de las capas iniciales. Además posee un skip connection (también llamado identity o shortcut) desde el inicio hasta el final del Bottleneck.

   Hint: Ninguna capa convolucional debe utilizar bias. **Ojo**, los BatchNorm deben ser 2D.

__init__()

(a) (1 punto) Cree la primera capa convolucional como `conv1` que vaya de `in_channels` a `out_channels`. El kernel debe ser 1x1, el stride 1 y el padding 0.

(b) (1 punto) Agregue un BatchNorm y un Relu, con los nombres bn1 y act1 respectivamente.

(c) (1 punto) Cree una segunda capa convolucional, en este caso que vaya de `out_channels` a `out_channels`. El kernel debe ser 3x3, el padding 1 y el stride debe recibirse de la variable `stride`. Esto corresponde al proceso de downsampling mencionado en la Figura 3 del paper.

(d) (1 punto) Agregue un BatchNorm y un Relu, con los nombres bn2 y act2 respectivamente.

(e) (1 punto) Cree una tercera capa convolucional, en este caso que vaya de `out_channels` a `expanded_channels`. El kernel debe ser 1x1, el stride 1 y el padding 0.

(f) (1 punto) Agregue un BatchNorm y un ReLU, con los nombres bn3 y act3 respectivamente.

(g) (1 punto) Agregue una variable `downsample` que reciba el parámetro `downsample` y una ReLU llamada relu. `downsample` corresponderá a una capa convolucional que corrige los tamaños entre los canales de la skip connection antes y después de la expansión de canales **(Esta capa la implementaremos más adelante)**.

forward()

(h) (2 puntos) Genere el forward pass. Para ello defina la skip connection (como identity) que pase a traves de la capa `downsample`. Luego el forward pass debe recorrer las 3 convoluciones y sus respectivos batchnorm y activation functions para luego volver a sumarse a la skip connection, y pasar por la última relu. Ver Figura 2 y Figura 5 del Paper.

2. Implemente la clase `Resnet()`.

downsample_layer()

(a) (1 punto) Esta función creará la capa convolucional que corrige los canales de nuestro skip connection. Cree una capa convolucional que vaya de `in_channels`, a `expanded_channels`, con kernel 1x1, y que reciba el stride. Debe ir seguida de un BatchNorm dentro de un bloque `Sequential`.

make_resnet_layers()

(b) (1 punto) Esta función creará los BottleNecks que, de acuerdo a la Tabla 1 del paper, deben repetirse de acuerdo a la arquitectura a utilizar. Cree la variable `downsample` la cual aplicará la función `downsample_layer()` recibiendo `self.in_channels` a `out_channels*4`, el stride provendrá de `downsample_stride`.

(c) (1 punto) Realice un append de un Bottleneck que vaya de `self.in_channels` a `out_channels`, debe recibir el `downsample` y el stride de `bottleneck_stride`. La Figura 3 muestra el por qué se aplica el downsample.

(d) (1 punto) Actualice el valor de `self.in_channels` a `out_channels*4`. Esto permitirá la compatibilidad de los siguientes Bottlenecks. Dentro de un for loop haga un append de los Bottlenecks restantes que vayan de `self.in_channels` a `out_channels`. El número de Bottlenecks está regido por la variable `num_res_layers`.

(e) (1 punto) La función debe retornar un bloque `Sequential` con todas las capas creadas.

__init__()

(f) (1 punto) Partiremos definiendo las capas de la primera parte de la red. Estas corresponden a las 2 primeras filas de la Tabla 1 del paper. Cree una primera capa convolucional con el nombre conv1. Debe ir de `image_channels` a `in_channels`. El `kernel_size` debe ser 7 y el stride 2. Utilizando la fórmula dada en clases cálcule cuánto debería ser el padding para obtener el output_size indicado en la Tabla 1 del paper.

(g) (1 punto) Agregue un BatchNorm y un Relu, con los nombres bn1 y act1 respectivamente.

(h) (1 punto) Agregue un MaxPool (con nombre maxpool) con Kernel 3x3 y stride 2. Utilizando la fórmula dada en clases calcule cuánto debería ser el padding para obtener el output_size indicado en la Tabla 1 del paper.

(i) (1 punto) Cree las capas layer1, layer2, layer3 y layer4 como Resnet layers. Cada capa debe recibir el número de layers y sus `out_channels` (que corresponden al número de canales de la primera capa de cada bottleneck). La layer1 tendrá strides de 1, el resto tendrá strides de 2.

(j) (1 punto) Genere un Average Pool (llamada `global_pool`) que garantice que la salida será siempre de 1x1.

(k) (1 punto) Aplique un Flatten y una capa Linear (no LazyLinear) (llamadas flatten y fc respectivamente) que lleve del número de dimensiones correspondientes hasta `num_classes`.

forward()

(l) (1 punto) Cree el forward pass pasando por cada una de las capas creadas en el orden correspondiente.

(m) (0 puntos) Verifique su implementación ejecutando el comando `python resnet.py`. El Script le indicará si su implementación calza o no con la implementación de **timm**. Una implementación exitosa implica que el número de parámetros debe coincidir. **¡¡Buena Suerte!!**

OJO: No se requiere entrenar este modelo.

Parte 2: Fine-Tuning

3. Entrene un modelo de Clasificación de Especies de Plantas. Para ello se entrega el siguiente **dataset**. **Ojo**: El archivo zip pesa casi 5GB, descárguelo con tiempo. Se recomienda trabajar en Colab, va a necesitar GPU.

   Ojo: Guarde el archivo **create_dataset.py** y el archivo **house_plant_species.zip** en la misma carpeta.

   Esta parte será en formato competencia. El grupo que reporte el mejor Accuracy en el Validation Set dado (siempre que sea de manera correcta) obtendrá el derecho de **borrar un control adicional**.

   (a) (1 punto) Ejecute `python create_dataset.py -p house_plant_species -tp 0.8`. Este script permitirá descomprimir el archivo además de crear carpetas con fotos de entrenamiento y validación. **No descomprima de manera manual**.

   (b) (3 puntos) Realice el proceso de Fine-Tuning de una arquitectura pre-entrenada de la librería **timm**. Puede escoger entre las arquitecturas mencionadas en clases.

   (c) (2 puntos) Aplique técnicas de Data Augmentation online (mientras se entrena el modelo) a su criterio.

   (d) (1 punto) Guarde un checkpoint de su mejor modelo obtenido. Reporte el Accuracy obtenido y todo el proceso de entrenamiento en un Jupyter Notebook. Utilizar código ordenado y modularizado como hemos visto en clases será recomenpensado. **Así que hágalo!!**

Parte 3: Deployment

Los modelos que viven en un Jupyter Notebook no son útiles. Para ello crearemos un Demo, de una aplicación **minimalística muy simple** utilizando Gradio.

(a) (3 puntos) Utilizando el checkpoint de la parte 2. Genere un archivo **app.py** que contenga una aplicación que reciba una imagen de input, y devuelva la clasificación de la Especie de Planta. Sólo se evaluará funcionalidad y no estética (**aunque puede ponerle cariño**).

(b) (5 puntos) La aplicación debe funcionar en vivo durante la defensa de código con imágenes que le entreguemos. (**Se evalúa la funcionalidad de la app y no la precisión de la predicción**).

Hint: Asegúrese de usar `.launch(share = True)`.