PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
IIC-3697 APRENDIZAJE PROFUNDO

TAREA 1: REDES NEURONALES CONVOLUCIONALES (CCNS)

Fecha de Entrega: 24 de Abril

Objetivo

Estimad@s, en esta actividad tendrán la oportunidad de poner en práctica sus conocimientos sobre aprendizaje profundo (deep learning). En particular, podrán experimentar con las técnicas que discutimos en clases para implementar Redes Neuronales Convolucionales (Convolutional Neural Nets o CNNs). Adicionalmente podrán ver en vivo y en directo el poder de las GPUs para acelerar el entrenamiento de redes de aprendizaje profundo.

1 Parte 1: VGG-19 (30%).

Para ilustrar la implementación de CNNs en Keras, estudiaremos en profundidad una de las estructuras más populares: VGG-19 [3]. La figura 1 muestra la estructura de VGG-19:

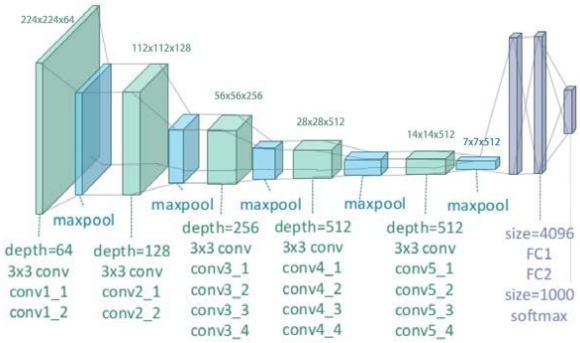

Figura 1: Arquitectura de la red convolucional VGG-19. Cada bloque (rectángulo verde + rectángulo azul) de la imagen corresponde a un *stack* de capas convolucionales y *MaxPooling*. La configuración de dichas capas convolucionales se encuentra abajo del bloque que las contienen. Por ejemplo, el primer *stack* de la red contiene dos capas de convolución (`conv1_1, conv1_2`) con 64 filtros de 3x3 que dejan a los datos de 224x224x64 y una capa de `MaxPooling` que reduce la dimensionalidad a 112x112x64.

Para implementar esta red en Keras, comenzaremos creando el contenedor (container) que será usado para encapsular el modelo:

```
modelVGG19 = Sequential()
```

Como comentamos en clases, para aprovechar de mejor forma la información en los bordes de la imagen de entrada, agregamos filas y columnas con ceros alrededor de ella (padding). En el caso de VGG-19 se agregan 1 fila y 1 columna a cada lado de la imagen de entrada. Para esto usamos la función de Keras `ZeroPadding2D`, según el siguiente formato:

```
modelVGG19.add(ZeroPadding2D((1,1),input_shape=(224, 224, 3)))
```

El primer parámetro indica el padding de 1 pixel en cada borde (horizontal y vertical), mientras que el segundo parámetro (`input_shape`) indica el tamaño de la imagen de entrada.

Con el contenedor y padding listos comenzamos a agregar las capas convolucionales del primer bloque. Estas capas contienen 64 filtros de 3x3, con un stride de 1 pixel en la dirección horizontal y 1 en la dirección vertical. Para esto usamos la función de Keras `Conv2D`, según la siguiente notación:

```
modelVGG19.add(Conv2D(64, (3, 3), strides=(1, 1)))
```

El primer parámetro indica la cantidad de filtros y el segundo el tamaño de cada filtro. El parámetro `strides=(1,1)` indica el tamaño del stride horizonal y vertical.

Como vimos en clases, generalmente después de las capas convolucionales viene una capa de activación. VGG-19 utiliza rectificadores lineales (Relu) como función de activación.

Para implementarlas en Keras usamos el módulo `Activation`, según la siguiente notación:

```
modelVGG19.add(Activation(activation="relu"))
```

Por motivos de simplicidad, Keras permite definir capas convolucionales con los parámetros (`activation`, `padding`) que agrupan estas tres capas en una. El parámetro (`padding='same'`) agrega filas y columnas con ceros para que pueda coincidir el filtro convolucional con el tamaño de la imágen. Por otra parte, El parámetro (`activation='relu'`) agrega una capa de activación del tipo Relu. Por lo tanto, las tres capas agregadas recien los podemos resumir en :

```
modelVGG19.add(Conv2D(64, (3, 3), input_shape=(224, 224, 3), strides=(1, 1),
    activation="relu", padding="same"))
```

Como muestra la figura 1, VGG-19 lleva dos capas convolucionales iguales, por lo que agregaremos otra igual a nuestro modelo. El último paso del primer bloque de VGG-19 consiste en la aplicación de un operador *Max-Pooling*. En este caso, el operador actúa sobre una vecindad de 2x2 utilizando un *stride* de 2 posiciones en las direcciones vertical y horizontal. Para esto usamos la función de Keras `MaxPooling2D`, según la siguiente notación:

```
modelVGG19.add(MaxPooling2D((2,2), strides=(2,2)))
```

Uniendo todo lo anterior y agregando la definición de las librería necesarias, obtenemos el siguiente código:

```
#Definicion de librerias con la funciones que seran utilizadas por Keras.
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Activation, Dropout,
    Flatten

#Definicion de contenedor y primera capa de VGG19.
modelVGG19 = Sequential()
modelVGG19.add(Conv2D(64, (3, 3), input_shape=(224, 224, 3), strides=(1, 1),
    activation="relu", padding="same"))
modelVGG19.add(Conv2D(64, (3, 3), strides=(1, 1), activation="relu", padding="
    same"))
modelVGG19.add(MaxPooling2D((2,2), strides=(2,2)))
```

El archivo `VGG19Capa1.py`, disponible en el sitio web del curso, contiene el código anterior. Ejecute este código y verifique que las dimensiones de salida de la última capa definida corresponden a las utilizadas por VGG–19. Para esto puede utilizar en Keras el comando: `print(modelVGG19.output_shape)`.

Este comando permite imprimir en pantalla la dimensiones de salida de la red definida hasta la ejecución del comando. A modo de ejemplo, para acceder a las dimensiones de salida de la red antes y después de aplicar el operador *Max-Pooling*, podemos ejecutar:

```
#Definicion de librerias con la funciones que seran utilizadas por Keras.
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Activation, Dropout,
    Flatten

#Definicion de contenedor y primera capa de VGG19.
modelVGG19 = Sequential()
modelVGG19.add(Conv2D(64, (3, 3), input_shape=(224, 224, 3), strides=(1, 1),
    activation="relu", padding="same"))
modelVGG19.add(Conv2D(64, (3, 3), strides=(1, 1), activation="relu", padding="
    same"))
print(modelVGG19.output_shape) # dims de la red antes de MaxPooling
modelVGG19.add(MaxPooling2D((2,2), strides=(2,2)))
print(modelVGG19.output_shape) # dims de la red despu\'es de MaxPooling
```

Siguiendo con la arquitectura de VGG19 en la figura 1 y las funciones definidas anteriormente, el segundo bloque queda definido por:

```
modelVGG19.add(Conv2D(128, (3, 3), strides=(1, 1), activation="relu", padding="
    same"))
modelVGG19.add(Conv2D(128, (3, 3), strides=(1, 1), activation="relu", padding="
    same"))
modelVGG19.add(MaxPooling2D((2,2), strides=(2,2)))
```

Actividad 1

Verifique que las dimensiones de salida de este segundo bloque corresponden a las utilizadas por VGG–19. En su informe de laboratorio incorpore los *output* generados. Adicionalmente, utilice el comando `modelVGG19.summary()` para generar un resumen de la red construida hasta este momento. ¿Cuántos parámetros (pesos) contiene esta red?

Actividad 2

Usando como guía las capas anteriores, construya en Keras el tercer bloque de VGG–19. Tenga presente que en este tercer bloque cambia la cantidad de capas convolucionales y la cantidad de filtros lleva cada capa convolucional.

En su informe de laboratorio reporte el código generado. Adicionalmente, utilice la función `output_shape` para verificar que la salida del tercer bloque corresponde a la arquitectura de la figura 1, y la función `summary` para obtener un resumen de los parámetros de la red.

Actividad 3

Para completar la fase convolucional de VGG–19 nos queda definir los bloques 4 y 5. Análogamente al bloque 3, los bloques 4 y 5 contienen una cantidad distinta de capas y filtros convolucionales que el bloque 2.

Genere el código de los bloques 4 y 5, verifique que las salidas sean las adecuadas, y determine el número de parámetros de la red. Reporte sus observaciones y código generado.

Actividad 4

Ha sido una tarea ardua pero estamos cerca de completar la implementación de VGG–19, sólo nos queda la definición de las capas de conexión densa (multilayer perceptron o MLP).

Para la definición de la primera de estas capas, el primer paso es llevar a una representación 1D la salida 3D de la capa 5 (representada en figura 1 con un cubo). Para esto utilizamos la función de Keras `Flatten` (aplanar), según la siguiente sintaxis:

```
modelVGG19.add(Flatten())
```

Utilice la función `output_shape` para verificar el efecto de la función `Flatten`. ¿Las dimensiones obtenidas corresponden a lo esperado?. Fundamente sus observaciones.

Actividad 5

Las capas densas son definidas en Keras mediante la función `Dense()`. Revise sus apuntes y genere el código apropiado para definir la primera capa densa que tiene como salida 4.096 neuronas. Tal como en las capas convolucionales, VGG–19 utiliza para la capa 6 una función de activación *Relu*.

Verifique que las salidas sean las adecuadas, y determine el número de parámetros de la red. Reporte sus observaciones y código generado.

Actividad 6

Defina las capas 7 y 8 de VGG–19. En el caso de la capa 7, como se aprecia en la figura 1, tiene una salida de 4.096 neuronas. Esta capa utiliza función de activación Relu. Finalmente, la capa 8 tiene una salida de 1.000 neuronas. Esta capa utiliza función de activación *softmax*.

En su reporte de laboratorio incorpore el códigos generado, así como un análisis del número de filtros y parámetros de la red final. ¿ Qué bloques utilizan más filtros y parámetros ?, ¿Qué justifica este tipo de arquitectura?. Comente y fundamente sus observaciones.

2 Parte 2: Reconocimiento de Visual (70%).

Una de las aplicaciones donde los modelos profundos convolucionales han tenido mayor éxito es el reconocimiento visual. En esta parte de la tarea se explorarán algunas arquitecturas que han surgido en este ámbito. En esta actividad tendrán la oportunidad de experimentar con 3 populares arquitecturas convolucionales: AlexNet [2], VGG-19 [3] y ResNet-50 [1].

Estos modelos se pueden encontrar implementados en el módulo `applications` de la librería Keras. Adicionalmente, es posible instanciar los modelos con pesos pre–entrenados sobre el conjunto de datos *ImageNet*.

Investigue sobre las redes AlexNet y ResNet–50, luego realice un breve cuadro que indique las principales diferencias entre estas redes y VGG–19. En particular, compare profundidad, número de parámetros y el rendimiento que estas redes obtienen en el conjunto de datos ImageNet. Comente sus observaciones principales.

2.1 mini–ImageNet

Para esta parte de la tarea deberán usar el conjunto de datos mini–ImageNet [5]. Este conjunto de datos es un subconjunto de ImageNet. Contiene 60.000 imágenes a color de 84 x 84 pixeles pertenecientes a 100 clases. Cada clase tiene 600 imágenes.

En el sitio web del curso se publicará un enlace para descargar este dataset.

Seleccione alguna de las redes que hemos estado discutiendo, vale decir AlexNet, VGG–19 o Resnet–50, y entrénela sobre el conjunto de datos mini–ImageNet. Para el entrenamiento de la red, defina y comente alguna estrategia de validación a fin de ajustar hiperparámetros, como el número de épocas a entrenar. Para la función de pérdida utilice *cross entropy* y para la última capa de la red utilice una función de activación *softmax*.

Actividad 7

Comente acerca de la estrategia de validación, tiempo de entrenamiento, ajuste de parámetros, algoritmo de optimización, tamaño de mini-batch .

Actividad 8

Usando la información de entrenamiento, realice gráficos que muestren:

- Evolución de la función de pérdida respecto de las épocas de entrenamiento.
- Evolución de la exactitud de clasificación sobre el set de entrenamiento y validación.
- Analice y comente los gráficos anteriores.

Actividad 9

Pruebe el modelo resultante en el set de test.

- Indique la exactitud promedio. Compare las exactitudes en el set de test, validación y entrenamiento. Comente.
- ¿Cuál es la clase con el mejor y cuál tiene el peor rendimiento?. Seleccione alguna imágenes de las clases con mejor y peor clasificación, inclúyalas en su informe y comente.

2.2 Conjunto de datos alternativo

Finalmente, deberán entrenar algunas de las redes discutidas anteriormente, es decir AlexNet, VGG–19 o Resnet–50, sobre algún conjunto de datos de libre elección. Se recomienda explorar conjuntos de datos que presenten tareas *ad hoc* a las redes convoluciones, como por ejemplo, clasificación de imágenes, reconocimiento de caras, etc.

Se pone a disposición un conjunto de datos denominado Places–25. Places–25 es un subconjunto del conjunto de datos Places–205 [4]. Contiene imágenes pertenecientes a 25 escenas. Cada escena cuenta con 5.000 imágenes.

En el sitio web del curso se publicará un enlace para descargar este dataset.

Se refuerza la idea de que idealmente deben buscar algún conjunto de datos que les parezca interesante para esta última parte. Si no encuentran ninguno, tienen la opción de usar Places–25.

Actividad 10

Describa el conjunto de datos utilizado. Se debe indicar número de imágenes, número de clases, número promedio de imágenes por cada clase y distribución del número de imágenes por cada clase. Además, insertar un par de imágenes de ejemplo con sus respectivas etiquetas.

Actividad 11

Comente acerca de la estrategia de validación, tiempo de entrenamiento, ajuste de parámetros, algoritmo de optimización, tamaño de mini-batch .

Actividad 12

Usando la información de entrenamiento, realice gráficos que muestren:

- Evolución de la función de pérdida respecto de las épocas de entrenamiento.
- Evolución de la exactitud de clasificación sobre el set de entrenamiento y validación.
- Analice y comente los gráficos anteriores.

Actividad 13

Pruebe el modelo resultante en el set de test.

- Indique la exactitud promedio. Compare las exactitudes en el set de test, validación y entrenamiento. Comente.
- ¿Cuál es la clase con el mejor y cuál tiene el peor rendimiento?. Seleccione alguna imágenes de las clases con mejor y peor clasificación, inclúyalas en su informe y comente.

Google Colaboratory

En la página web del curso podrán encontrar un documento con instrucciones para poder utilizar el sitio de Google: *Colaboratory*. Éste es un proyecto de Google orientado a diseminar la educación y la investigación en el área de aprendizaje de máquina, y nos permitirá utilizar GPUs para ejecutar las tareas del curso.

Consideraciones y formato de entrega

La tarea deberá ser entregada via SIDING en un cuestionario que se habilitará oportunamente. Se deberá desarrollar la tarea en un Jupyter Notebook con todas las celdas ejecutadas, es decir, no se debe borrar el resultado de las celdas antes de entregar. Si las celdas se encuentran vacías, se asumirá que la celda no fue ejecutada. Es importante que todas las actividades tengan respuestas explícitas, es decir, no basta con el *output* de una celda para responder.

Bibliografía

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*.

[2] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[3] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICRL*, 2015.

[4] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba and A. Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.

[5] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu and D. Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.