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

TAREA 1: REDES NEURONALES CONVOLUCIONALES (CCNS)

Fecha de Entrega: 30 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: AlexNet (35 %).

Para ilustrar la implementación de CNNs en Keras, estudiaremos en profundidad una de las estructuras más populares: AlexNet [2]. La figura 1 muestra la estructura de AlexNet:

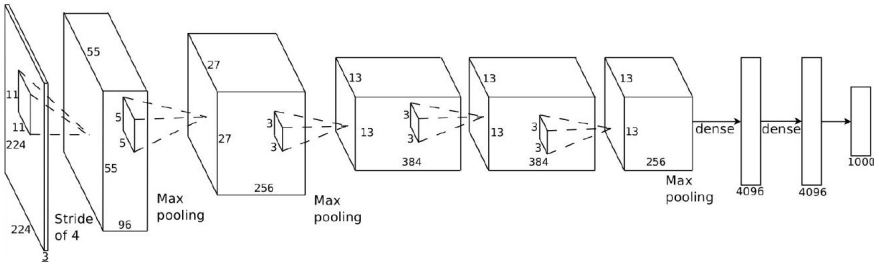

Figura 1: Arquitectura de la red convolucional AlexNet.

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

```
modelAlexNet = 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 AlexNet se agregan 2 filas y 2 columnas a cada lado de la imagen de entrada. Para esto usamos la función de Keras *ZeroPadding2D*, según el siguiente formato:

```
modelAlexNet.add(ZeroPadding2D((2,2),input_shape=(224, 224, 3)))
```

El primer parámetro indica el padding de 2 pixeles en cada borde (horizontal y vertical), mientras que el segundo parámetro (*input_shape*) indica el tamaño de la imagen de entrada.

Una vez que tenemos lista la entrada podemos definir el operador de convolución de la primera capa. En este caso corresponde a 96 filtros de 11x11, con un stride de 4 pixeles en la dirección horizontal y 4 en la dirección vertical. Para esto usamos la función de Keras *Convolution2D*, según la siguiente notación:

```
modelAlexNet.add(Convolution2D(96,(11,11), strides=(4,4), padding='valid'))
```

El parámetro *strides=(4,4)* indica el tamaño del stride horizonal y vertical. Por su parte, el parámetro *padding='valid'* indica que el barrido de la convolución se aplica hasta la última posición válida antes de superar el borde de la imagen.

Como discutimos en clases, en sus capas convolucionales AlexNet utiliza rectificadores lineales (Relu) como función de activación. Adicionalmente, aplica una normalización sobre las activaciones de las neuronas sobre cada batch de entrenamiento. Esta normalización se encarga de generar batch con salidas de media=0 y desviación standard=1. Este tipo de normalización es usual en CNNs y Keras la implementa mediante la función *BatchNormalization()*. En Keras, las 2 tareas anteriores quedan dadas por:

```
modelAlexNet.add(Activation(activation='relu '))
modelAlexNet.add(BatchNormalization())
```

Como muestra la figura 1, el último paso de la primera capa de AlexNet consiste en la aplicación de un operador *Max-Pooling*. En este caso, el operador actúa sobre una vecindad de 3x3 utilizando un stride de 2 posiciones en las direcciones vertical y horizontal. Para esto usamos la función de Keras *MfdaxPooling2D*, según la siguiente notación:

```
modelAlexNet.add(MaxPooling2D((3,3), 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.layers import Activation, Dense, Flatten, Dropout
from keras.models import Sequential
from keras.layers.convolutional import Convolution2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D

#Definicion de contenedor y primera capa de AlexNet.
modelAlexNet = Sequential()
modelAlexNet.add(ZeroPadding2D((2,2), input_shape=(224, 224, 3)))
modelAlexNet.add(Convolution2D(96, (11,11), strides=(4,4), padding='valid'))
modelAlexNet.add(Activation(activation='rel'))
modelAlexNet.add(BatchNormalization())
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
```

El archivo AlexNetCapa1.py, disponible en el sitio web del curso, contiene el código anterior. Ejecute este código y verifique que las dimensiones de salida de la capa definida corresponden a las utilizadas por AlexNet. Para esto puede utilizar en Keras el comando: *print(modelAlexNet.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 contenedor y primera capa de AlexNet.
modelAlexNet = Sequential()
modelAlexNet.add(ZeroPadding2D((2,2), input_shape=(224, 224, 3)))
modelAlexNet.add(Convolution2D(96, (11,11), strides=(4,4), padding='valid'))
modelAlexNet.add(Activation(activation='relu '))
modelAlexNet.add(BatchNormalization())
print(modelAlexNet.output_shape) #dims de la red antes de max-pooling
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
print(modelAlexNet.output_shape) #dims de la red despues de max-pooling
```

Siguiendo con la arquitectura de AlexNet en la figura 1 y las funciones definidas anteriormente, la segunda capa queda definida por:

```
modelAlexNet.add(ZeroPadding2D((2,2)))
modelAlexNet.add(Convolution2D(256, (5, 5), padding='valid'))
modelAlexNet.add(Activation(activation='relu '))
modelAlexNet.add(BatchNormalization())
modelAlexNet.add(MaxPooling2D((3,3), strides=(2,2)))
```

Actividad 1

Verifique que las dimensiones de salida de esta segunda capa corresponden a las utilizadas por AlexNet. En su informe de laboratorio incorpore los output generados. Adicionalmente, utilice el comando *modelAlexNet.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 la tercera capa de AlexNet. Tenga presente que esta tercera capa:

- Incluye un padding de 1 cero a cada lado del mapa de activaciones de entrada.
- No utiliza normalización batch.
- No incorpora una etapa de max-pooling.

En su informe de laboratorio reporte el código generado. Adicionalmente, utilice la función *output_shape* para verificar que la salida de la tercera capa 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 AlexNet nos queda definir las capas 4 y 5. Análogamente a la capa 3, la capa 4 también realiza un padding de 1 cero a cada lado de la entrada, no incluye normalización batch, y no incluye max-pooling. Por su parte, la capa 5 realiza un padding de 1 cero a cada lado de la entrada, no incorpora normalización batch, pero si incorpora una etapa de max-pooling con una ventana de 3x3 y un stride de 2 en las direcciones vertical y horizontal.

Genere el código de las capas 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 AlexNet, 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:

```
modelAlexNet.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

Como vimos en clases, 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 4096 neuronas. Tal como en las capas convolucionales, AlexNet utiliza para la capa 6 una función de activación *Relu*. Adicionalmente, incorpora un proceso de *Dropout* con una probabilidad de 0.5.

Revise sus apuntes de clases, donde se ilustra el uso de Dropout en Keras, y genere el código restante para completar la definición de la capa 6 de AlexNet. 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 AlexNet. En el caso de la capa 7, como se aprecia en la figura 1, tiene una salida de 4096 neuronas. Esta capa utiliza función de activación Relu y Dropout con probabilidad 0.5. Finalmente, la capa 8 tiene una salida de 1000 neuronas. Esta capa utiliza función de activación *softmax* y no utiliza Dropout.

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

Actividad 7

En esta parte tendrán la oportunidad de probar la red AlexNet. Para ello utilizarán el set de datos MIT-10, el cual contiene imágenes de 10 tipos de escenas. Este set está disponible en el siguiente link: `https://www.dropbox.com/s/c0d7npcg1yysnux/MIT-10-Classes.zip?dl=0`. En el directorio Feats, este set contiene las 4096 features correspondientes a la salida del layer 6 de AlexNet para cada una de las imágenes del set. Utilice estas features para entrenar un clasificador del tipo SVM o MLP (multilayer perceptron, i.e., red neuronal tipo feedforward).

Entrene utilizando las features en el subdirectorio TrainSet y pruebe el rendimiento del clasificador resultante utilizando las features en el directorio TestSet. Reporte la exactitud final de clasificación en ambos set de datos.

Actividad 8

Repita el ejercicio anterior, pero en esta oportunidad en lugar de usar las features precalculadas de AlexNet, entrene esta red con el set MIT-10. Reporte sus resultados indicando los siguientes gráficos:

- Evolución de la función de pérdida respecto de las épocas de entrenamiento.
- Evolución de la exactitud de clasificación sobre el set de entrenamiento y test, respecto de las épocas de entrenamiento.
- Analice y comente los gráficos anteriores, sea breve y preciso.

2. Parte 2: Reconocimiento de Visual (65 %).

Una de las aplicaciones donde los modelos tipo CNN han tenido mayor éxito es el reconocimiento visual. En esta parte de la tarea exploraremos algunas arquitecturas que han surgido en este ámbito. En particular, en esta actividad tendrán la oportunidad de experimentar con 2 populares arquitecturas convolucionales: VGG-16 [3] y ResNet-50 [1]. Ambas redes son parte de las librerías de Keras, es decir, sus estructuras han sido previamente codificadas en Keras y entrenadas con el set de datos ImageNet. A modo de ejemplo, la figura 1 muestra la estructura de VGG-16:

Actividad 9

Investigue sobre las redes VGG-16 y ResNet-50, luego realice un breve cuadro que indique las principales diferencias entre estas redes y AlexNet. En particular, compare profundidad, número de parámetros, y el rendimiento que estas redes obtienen en el set ImageNet. Comente brevemente sus observaciones.

En la página web del curso podrá encontrar un documento con instrucciones para utilizar las herramientas cloud del sitio: *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

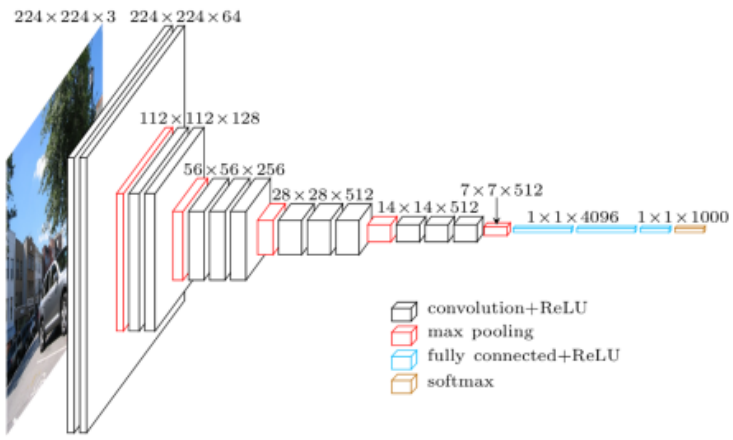

Figura 2: Arquitectura de la red convolucional VGG-16.

del curso. Adicionalmente, en el sitio web del curso encontrará instrucciones sobre 2 set de datos. Debe seleccionar sólo uno de estos set de datos y utilizarlo para realizar todas las actividades siguientes.

Actividad 10

Utilizando *Google-Colaboratory* modifique y entrene algunas de las redes que hemos estado discutiendo, vale decir: AlexNet, VGG16 o Resnet-50. Para este entrenamiento parta desde cero, es decir, inicialice los pesos de la red utilizando valores aleatorios. Utilice el set de validación proporcionado para ajustar meta-parámetros tales como el número de épocas. Para la función de pérdida utilice cross entropy y para la última capa de la red utilice una función soft-max. En su reporte, comente acerca de como modificó el modelo original. También comente acerca del tiempo de entrenamiento, ajuste de parámetros, algoritmo de optimización y tamaño de mini-batch usados.

Actividad 11

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, respecto de las épocas de entrenamiento.
- Analice y comente los gráficos anteriores, sea breve y preciso.

Actividad 12

Pruebe el modelo resultante en el set de test. Indique la exactitud promedio del modelo final sobre los set de entrenamiento, validación y test. Comente sus resultados, sea breve y preciso.
Indique la clase con el mejor y la clase el peor rendimiento. Seleccione algunas imágenes de estas 2 clases e inclúyalas en su informe. Comente sus observaciones, sea breve y preciso.

Bibliografía

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.