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

TAREA 1: REDES NEURONALES CONVOLUCIONALES (CCNS)

Fecha Máxima de Entrega: 9 de Mayo, 23:59

Objetivo

Estimad@s, en esta tarea 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 Networks o CNNs). Adicionalmente, podrán experimentar con Google-Colab y observar el poder de las GPUs para acelerar el entrenamiento de redes de aprendizaje profundo.

1. Parte 1: ResNet-50 (20 %).

Para ilustrar la implementación de CNNs en Keras, estudiaremos en profundidad una de las estructuras más populares del 2015: Resnet-50 [1]. La figura 1 y el cuadro 1 muestran la arquitectura de la red. En este modelo, luego de cada capa convolucional se utiliza batch normalization (lo veremos en clase) y funciones de activación tipo ReLU.

Para implementar esta red en Keras, comenzaremos creando las primeras capas del modelo:

```
x = Input(shape=(224,224,3))
x= Conv2D(64, (7, 7), strides=(2, 2), name="conv1")(x)
x = BatchNormalization(name="bn_conv1")(x)
x = Activation("relu")(x)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)
```

Una vez definidas las primeras capas, procedemos a agregar la primera etapa de bloques residuales:

```
def identity_block(input, filters, block, stage):

    name = f"block_{block}_stage_{stage}"

    f1, f2, f3 = filters

    x = Conv2D(# complete el codigo
    x = BatchNormalization(name = "bn1_" + name)(x)
    x = Activation("relu")(x)


    x = Conv2D(# complete el codigo
    x = BatchNormalization(name = "bn2_" + name)(x)
    x = Activation("relu")(x)

    x = Conv2D(# complete el codigo
    x = BatchNormalization(name = "bn3_" + name)(x)

    if # complete el codigo:
```

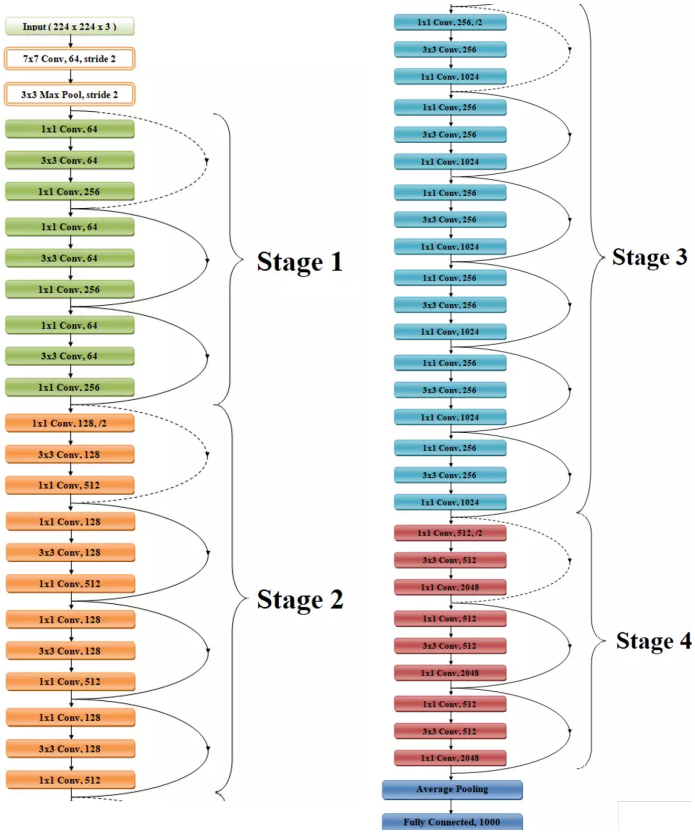


Figura 1: Arquitectura ResNet-50

```
        input = Conv2D(# complete el codigo

    x = Add()([x, input])
    x = Activation("relu")(x)

    return x
x = identity_block(x, [64,64,256], 1, 1)
x = identity_block(x, [64,64,256], 1, 2)
x = identity_block(x, [64,64,256], 1, 3)
x = identity_block(x, [64,64,256], 1, 4)
```

Actividad 1

Investigue y explique en **no más de tres líneas** la utilizad de una capa de convolución con kernel de 1x1.

Actividad 2

ResNet utiliza los denominados bloques residuales, según muestra la figura 2.

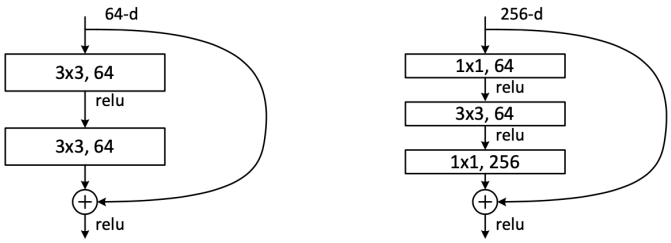


Figura 2: Bloque residual

Investigue e indique en no más de 5 líneas la teoría, funcionamiento y utilidad de estos bloques.

Actividad 3

Complete el código presentado anteriormente e indique las dimensiones del tensor resultante a la salida de la primera etapa de bloques residuales.

Actividad 4

Basándose en la arquitectura de la red, modele las etapas 2, 3 y 4, e indique las dimensiones del tensor resultante a la salida de cada bloque residual.

Actividad 5

Implemente la etapa final de clasificación de utilizando en la capa final una función de activación del tipo Softmax.

Actividad 6

Implemente una función para generar bloques residuales. Utilizando esta función instancie el modelo de la red. Luego, verifique la dimensionalidad de las capas. Reporte su función.

2. Parte 2: Análisis de redes (40 %).

En esta sección exploraremos técnicas de visualización en redes neuronales, tanto de sus representaciones internas (feature visualization), como de las partes del input que son responsables del output (attribution). Para este propósito utilizaremos las librerías Lucent y PyTorch.

Pytorch nos permite instanciar modelos como Alexnet[2], VGG[3], ResNet o Inception[4], junto con la posibilidad de descargar pesos pre-entrenados utilizando el dataset ImageNet. Para ello debemos utilizar torchvision, una libería de pytorch enfocada en el trabajo con imágenes.

```
import torch
import torchvision.models as models

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

alexnet = models.alexnet(pretrained=True).to(device).eval()

googlenet = models.googlenet(pretrained=True).to(device).eval()

vgg19 = models.vgg19(pretrained=True).to(device).eval()
resnet50 = models.resnet50(pretrained=True).to(device).eval()
```

Ahora que tenemos nuestros modelos en Pytorch, visualizaremos sus representaciones internas en distintas capas. Para ellos utilizaremos la librería torch-lucent y matplotlib.

```
!pip install torch-lucent
!pip install -U matplotlib

from lucent.optvis import render, param, transform, objectives
from lucent.modelzoo.util import get_model_layers
```

A modo de ejemplo utilizaremos AlexNet, cuya arquitectura se presenta en la figura 3.

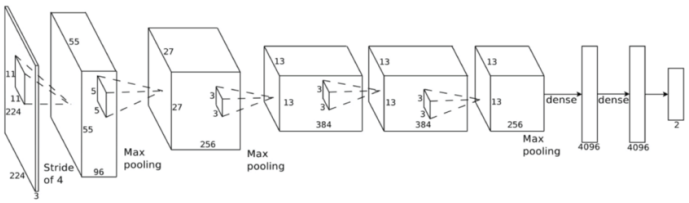


Figura 3: Arquitectura AlexNet

Primero debemos ver los elementos que conforman la red importada desde PyTorch.

```
get_model_layers(alexnet)
```

```
["features",
 "features_0",
 "features_1",
 "features_2",
 "features_3",
 "features_4",
 "features_5",
 "features_6",
 "features_7",
 "features_8",
 "features_9",
```

```
"features_10",
"features_11",
"features_12",
"avgpool",
"classifier",
"classifier_0",
"classifier_1",
"classifier_2",
"classifier_3",
"classifier_4",
"classifier_5",
"classifier_6"]
```

El output nos indica que el modelo posee 12 capas de features, sin embargo, AlexNet posee 5 capas convolucionales. Lo que sucede es que el modelo implementado se ve como:

```
class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )
```

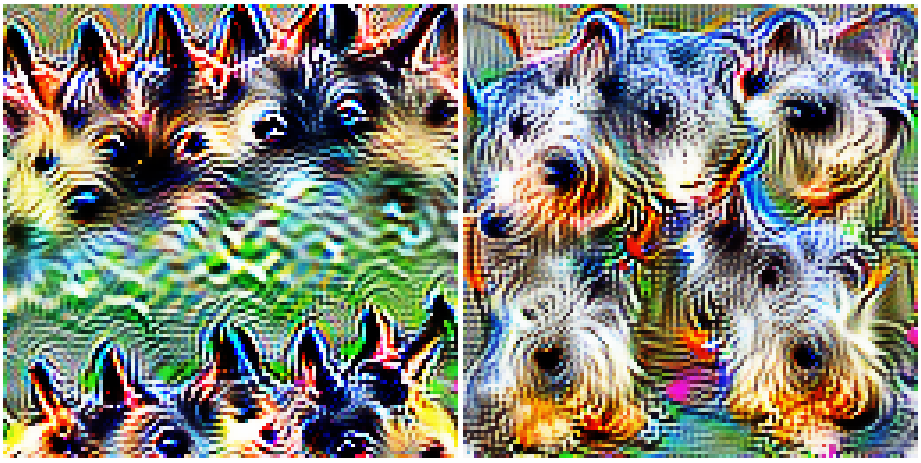
Como podemos ver, el atributo self.features corresponde a un nn.Sequential con 13 elementos. Como estos elementos no tienen nombre, se les asigna el nombre del atributo más un sufijo. Así, features_0 corresponde al primer elemento de nn.Sequential (nn.Conv2d) y features_12 corresponde al último elemento de nn.Sequential (nn.MaxPool2d). De esta manera, para referirnos a cada una de las 5 capas convolucionales, debemos considerar features_1, features_4, features_7, features_9 y features_11, ya que corresponden a la salida de la respectiva convolución después de la aplicación de la función de activación.

Ahora que ya sabemos cómo acceder a las capas que nos interesan de cada modelo, veamos cómo usar la librería. La función principal de Lucent pertenece al módulo render y se llama render_vis. Sus parámetros más importantes son una instancia del modelo que queremos visualizar y el canal que nos interesa. El canal se entrega como un string con formato nombre_de_la_capa:número_del_canal. Veamos un ejemplo visualizando el canal 68 de la capa 5 de AlexNet:

```
_ = render.render_vis(alexnet, 'features_11:68', show_inline=True) # show_inline
    =True despliega la imagen encontrada
```

También podemos visualizar una imagen correspondiente a alguna de las clases del set de datos. Para ello usamos el formato labels: índice_de_la_clase.

```
_ = render.render_vis(alexnet, 'labels:196', show_inline=True)
```



((a)) Respresentación features ((b)) Representación Labels

Figura 4: Renders Lucent.

Actividad 7

Seleccione una red entre VGG, Resnet e Inception y reporte la visualización de 3 canales (filtros) para mapas de activación correspondientes a capas iniciales, intermedias y finales de la red seleccionada. Puede ver las implementaciones de las redes en link. Comente brevemente sus impresiones sobre las visualizaciones obtenidas.

Actividad 8

Seleccione cualquiera de las redes preentrenadas disponibles en torchvision y modifique el head de clasificación para se que sea compatible con el dataset de flores disponible en link [5]. Utilizando los pesos preentrenados como punto de partida, re-entrene la red (fine-tuning) para clasificar los datos del set de flores. Reporte sus resultados a través de los siguientes gráficos:

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

Actividad 9

Para su modelo de clasificación de flores, utilizando Lucent, compare las features de una capa inicial, una capa intermedia y una capa final. También analice tres labels y compárelos con las imágenes de la clase correspondiente. Comente brevemente.

3. Parte 3: Detección de objetos (40 %).

A lo largo de esta tarea han experimentado con modelos de clasificación de imágenes, analizado su aprendizaje y experimentado con entrenamiento y fine-tuning. En esta sección pondrán en práctica lo que vimos en clases sobre técnicas de reconocimiento de objetos.

Para este propósito utilizaremos la red Faster R-CNN disponible en torchvision.

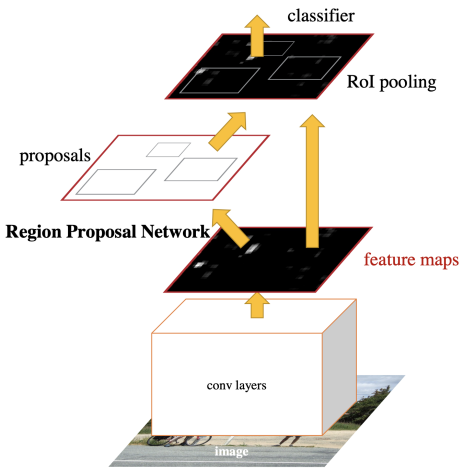


Figura 5: Faster R-CNN

La librería considera el uso de varios modelos base (backbones), en esta actividad ocuparemos ResNet-50 Faster R-CNN. Para instanciar este modelo preentrenado se debe ejecutar un código similar al siguiente.

```
from torchvision.models.detection import fasterrcnn_resnet50_fpn
frcnn_model = fasterrcnn_resnet50_fpn(pretrained=True, progress=True,
    num_classes=91, pretrained_backbone=True, trainable_backbone_layers=3)
```

Actividad 10

Investigue y explique, en no más de 10 líneas, el concepto de non-maximum suppresion en el contexto de reconocimiento de objetos. Adicionalmente, instancie e imprima el modelo para visualizar sus componentes. Explique en qué consisten los siguientes componentes (no más de 5 líneas c/u): Backbone, Feature Pyramid Network (FPN) y RPN.

Actividad 11

Programe dos funciones para hacer inferencia:
i) Función que recibe como entrada el modelo, una imagen y un umbral. Luego retorna los bounding box, clase y score de las detecciones de objetos en la imagen de entrada.
ii) Función para visualizar en la imagen utilizada los bounding boxes, etiquetas y scores de las detecciones.

Utilice las funciones anteriores para realizar inferencia de tres imágenes que contengan objetos pertenecientes al dataset con el cual fue entrenado el modelo: COCO 2017[6].

Actividad 12

Cambie la estructura del clasificador (head) del modelo fasterrcnn_resnet50_fpn para que permita clasificar imágenes de mapaches (el head solo debe realizar esta tarea). Para este propósito utilice el Raccoon Detector Dataset [7]. Para su entrenamiento utilice una estrategia de fine-tunning en base al modelo pre-entrenado en COCO 2017. Realice gráficos de la evolución de las métricas true-positives, false-negatives, false-positives, y precision-recall durante el entrenamiento y verifique el rendimiento en imágenes del set de test.

4. Google Colaboratory

Para la ejecución de sus tareas utilice la plataforma *Google Colaboratory* (Colab). En la página web del curso podrán encontrar un documento con instrucciones de como utlizar esta herramienta. Colab 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.

5. Consideraciones y Formato de Entrega

La tarea debe ser entregada via SIDING en un cuestionario que se habilitará oportunamente. Se debe desarrollar la tarea utilizando 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.

El día miercoles 28/04 a las 17:00 hrs se realizará una ayudantía de la tarea. El link se avisará oportunamente. Adicionalmente, pueden usar el foro del curso en Canvas para realizar consultas.

6. Anexos

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Cuadro 1: Arquitecturas ResNet

Referencias

[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] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. Going deeper with convolutions. In CVPR 2015.

[5] https://www.robots.ox.ac.uk/ vgg/data/flowers/102/index.html

[6] https://cocodataset.org/home

[7] https://github.com/bing0037/Raccoon_dataset