# Primer practica: red ALEXNET

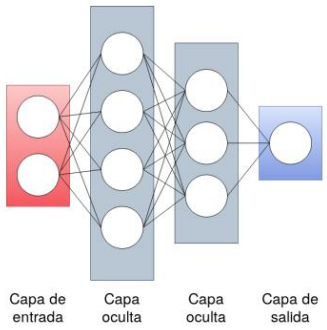
## Introducción

El aprendizaje profundo es un tipo de aprendizaje automatizado, en el cual se utilizan redes neuronales artificiales con una jerarquía establecida, donde en cada nivel se extraen las características significativas del nivel anterior. A esta jerarquía se le conoce como modelo neuronal.

Las redes neuronales artificiales son un modelo computacional que trata de emular los procesos biológicos del sistema nervioso, como la forma en la que el cerebro humano almacena información, aprende y reconoce patrones. Están formadas por un de nodos conectados entre sí, cuyo principal objetivo es almacenar información, procesarla, y dependiendo de su estado de activación, transmitir o no señales a otros nodos.

La estructura general de una red neuronal artificial viene dada por conjuntos de capas de nodos, donde cada nodo representa una neurona artificial. Cada capa tiene las siguientes características:

- En la capa de entrada, se adquieren las señales de entrada, es decir, la información proveniente de sensores o de algún sistema de procesamiento de bajo nivel.
- En las capas ocultas, se extraen características abstractas de la señal de entrada y se obtienen los descriptores o los patrones más significativos de ´esta.
- En la capa de salida, se obtiene una clasificación, detección o interpretación de la señal de entrada.




## Desarrollo

Explicaremos cómo puede detectar objetos de transmisiones en vivo, como cámaras web, utilizando AlexNet en Python para la detección de objetos.

Primero necesitaremos instalar las siguientes bibliotecas:

1. La biblioteca `alexnet_pytorch` que implementa el algoritmo `AlexNet`
2. La biblioteca openCV para procesamiento de imágenes
3. La biblioteca NumPy se usa para almacenar datos de cada cuadro de nuestra cámara web en una matriz

4. importa la clase AlexNet

```
from alexnet_pytorch import AlexNet
```

5. crea un objeto de la clase de detección de objetos

```
model = AlexNet.from_pretrained("alexnet")
```

6. Se importan las 1000 etiquetas de predicción

```
labels_map = json.load(open("labels_map.txt"))

labels_map = [labels_map[str(i)] for i in range(1000)]
```

7. Se define una función de preprocesamiento para la imagen que captura la webcam antes de introducirla en la AlexNet previamente entrenada.
   - transforms.Resize((224,224)) : la entrada cambia de tamaño a 224x224
   - transforms.ToTensor() : coloque la entrada en formato Tensor
   - transforms.Normalize([media], [desviación estándar]): normaliza la entrada usando la [media] y la [desviación estándar]

```
def process_image(input_image):
        # Preprocess image
        preprocess = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
        input_tensor = preprocess(input_image)
        input_batch = input_tensor.unsqueeze(0)
        return input_tensor, input_batch
```

8. Cargar la imagen a la red pre entrenada desde la webcam para preprocesarla. El método de la biblioteca OpenCV VideoCapture() se usa para capturar transmisiones en vivo de cámaras conectadas a la computadora. Se debe definir la altura y el ancho del marco que mostrará los objetos detectados desde su transmisión en vivo.

```
import cv2

cam_feed = cv2.VideoCapture(0)

cam_feed.set(cv2.CAP_PROP_FRAME_WIDTH, 650)

cam_feed.set(cv2.CAP_PROP_FRAME_HEIGHT, 750)
```

9. El siguiente paso es cargar el modelo real junto con la imagen preprocesada, y regresa la etiqueta de clasificación y su porcentaje de probabilidad.

```
while True:
```

```
        start = time.time()
        ret, img = cam_feed.read()
# El cuadro capturado luego se pasa a la funcion process_image() para retornar en formato tensor
        input_tensor, input_batch = process_image(img)
# Después se mueve la entrada y el modelo a GPU si está disponible.
        if torch.cuda.is_available():
           input_batch = input_batch.to("cuda")
           model.to("cuda")
        with torch.no_grad():
           logits = model(input_batch)
# a partir de la variable logit se e9xtraen las 3 primeras predicciones sin el porcentaje de probabilidad
        preds = torch.topk(logits, k=3).indices.squeeze(0).tolist()
# Se extraen las tres primeras etiquetas con su porcentaje de probabilidad
        label0 = labels_map[preds[0]]
        prob0 = torch.softmax(logits, dim=1)[0, preds[0]].item()
        label1 = labels_map[preds[1]]
        prob1 = torch.softmax(logits, dim=1)[0, preds[1]].item()
        label2 = labels_map[preds[2]]
        prob2 = torch.softmax(logits, dim=1)[0, preds[2]].item()
        cv2.imshow("",img)
        listname = [[label0],[round(prob0,2)],[label1],[round(prob1,2)],[label2],[round(prob2,2)]]
# Cada 25 cuadros por segundo se iteran las 3 primeras predicciones para imprimirlo en la consola
        for x in itertools.zip_longest(*listname, fillvalue="-"):
            print('|\t|'.join([str(e) for e in x]))
            time.sleep(max(1./25 - (time.time() - start), 0))
# El proceso continúa hasta que presione la tecla "q" o "ESC" en su teclado
        if (cv2.waitKey(1) & 0xFF == ord("q")) or (cv2.waitKey(1)==27):
```

```
        break
cam_feed.release()
cv2.destroyAllWindows()
```

## Conclusiones

Para realizar esta práctica, encontré la implementación de Alex net del artículo de “ImageNet Classification with Deep Convolutional Neural Networks” para python y Pytorch. El objetivo de esta implementación es que sea simple, altamente extensible y fácil de integrar a otros proyectos. Hasta ahora se puede cargar los pesos sinápticos pre entrenados y usar los modelos para clasificación y extracción de características.

AlexNet compitió en el Desafío de reconocimiento visual a gran escala de ImageNet el 30 de septiembre de 2012. La red logró un error entre los 5 primeros del 15,3 %, más de 10,8 puntos porcentuales menos que el finalista. El resultado principal del documento original fue que la profundidad del modelo era esencial para su alto rendimiento, lo cual era computacionalmente costoso, pero factible debido a la utilización de unidades de procesamiento de gráficos (GPU) durante el entrenamiento.

3 48

A partir de esta nueva arquitecura se marco un nuevo referente en las redes neuronales profundas, ya que se alimenta de imágenes de alta calidad, y se validaron diferentes tecnicas para evitar el sobre entrenamiento de las neuronas como por ejemplo Max Pooling para crear un mapa de caracteristicas con una reduccion en el muestreo, y el dropout para deshabilitar neuronas de manera aleatoria para explorar nuevas conexiones entre neuronas.

Actualmente se cuentan con redes que pueden procesar imágenes con una resilucion más alta, con menos recursos computaciones, y esta red fue el arteaguas en el estado del arte.

## Créditos


- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90.
- https://pypi.org/project/alexnet-pytorch/#description