**INSTITUTO TECNÓLOGICO DE LA PAZ**

**DIVISIÓN DE ESTUDIOS PROFESIONALES**

**REPORTE SEMANAL DE RESIDENCIA PROFESIONAL**

**Semana 1**

**“Reconocimiento y seguimiento de objetos en entornos controlados”**

**CARRERA**

**Ingeniería en Sistemas Computacionales**

**PRESENTA**

**17310793 – Eloy Antonio Clemente Rosas**

**La Paz, Baja California Sur, México, 28 de Enero del 2021**

# Introducción

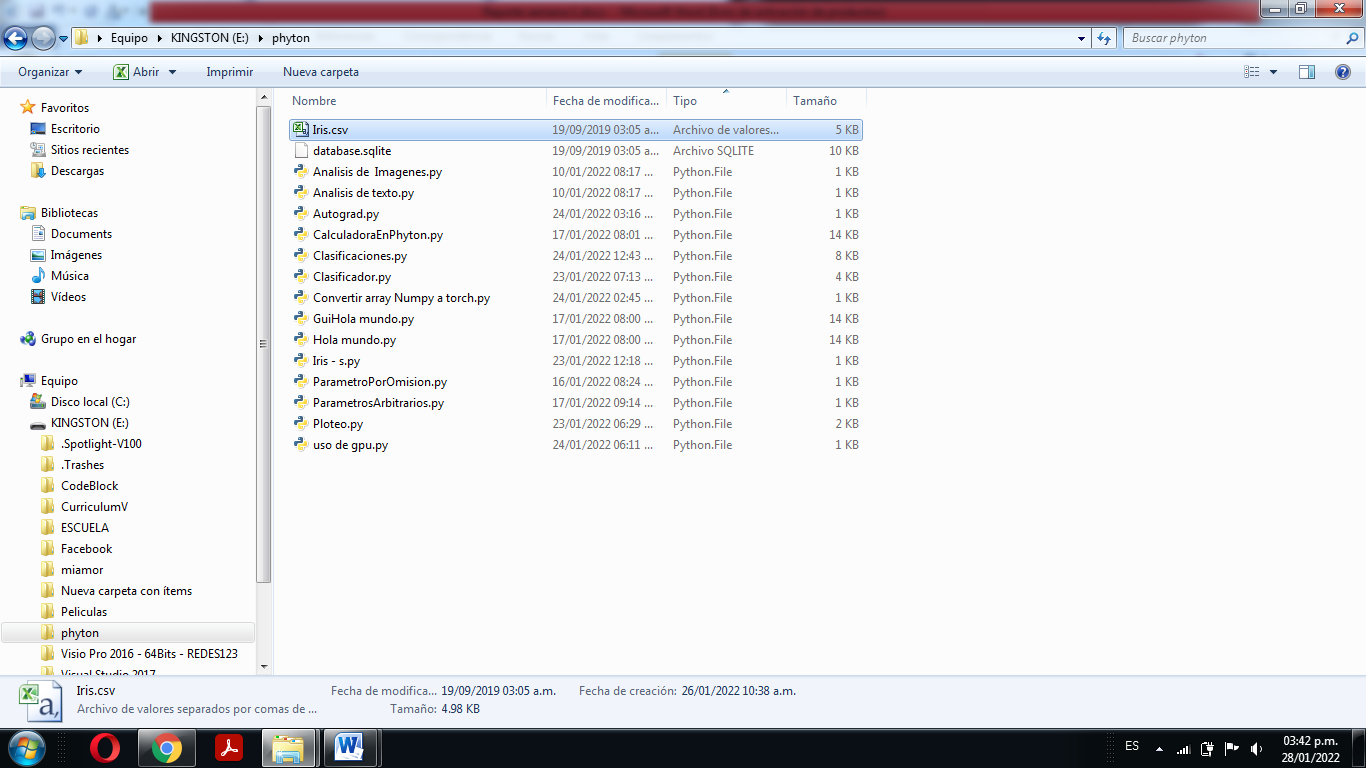
El presente documento contiene las actividades, y aprendizajes realizados durante la primera semana en mi residencia profesional. La cual fue una introducción teórica a los conocimientos y habilidades necesarias para el reconocimiento de objetos. En el documento se encuentran los conceptos obtenidos durante el desarrollo de las actividades, las librerías utilizadas durante el proceso de la programación, y el código para el entrenamiento de la red neuronal explicado.

Esta semana se centró en el aprendizaje del lenguaje de programación Python dentro de los 2 entornos de programación que son Spyder y Anaconda. Además de conocer los conceptos básicos del funcionamiento de un perceptron, una red neuronal artificial, su representación e interpretación en código, además, la librería de redes neuronales pytorch, y sus ventajas contra tensor flow.

# Desarrollo de actividades

El reto de la semana fue tener un clasificador de imágenes.

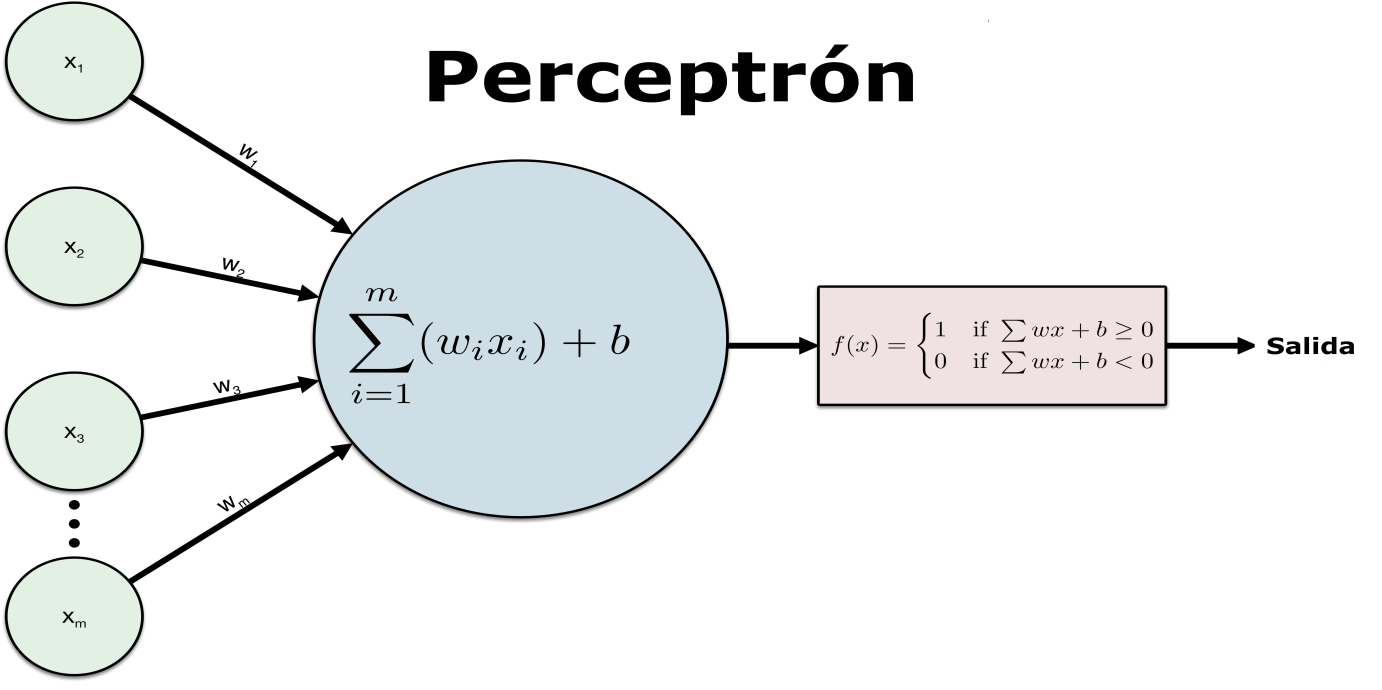
La primera actividad que realice durante la semana fue ponerme al tanto con el lenguaje de programación Python. Para esto realice algunos programas sencillos:

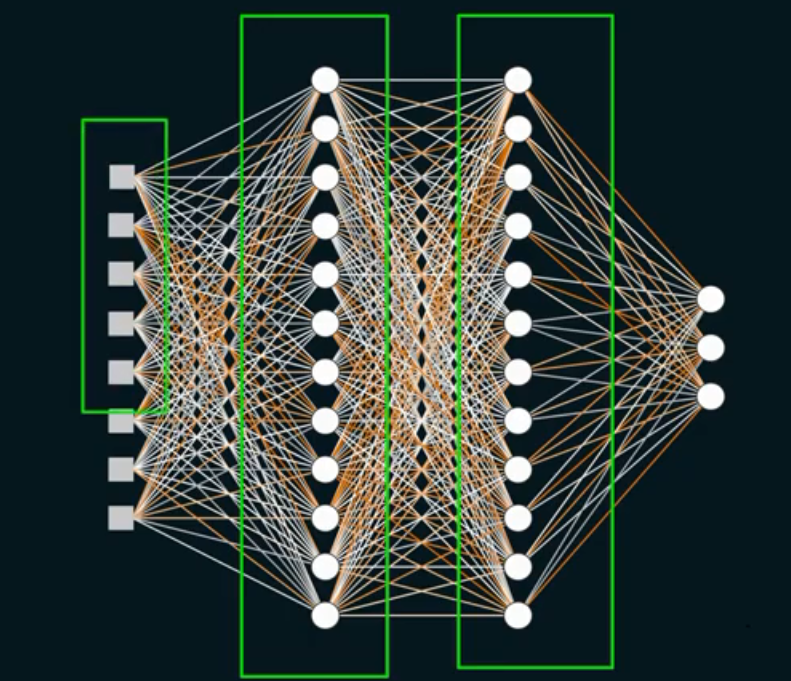


# Conceptos básicos

Es importante definir como se realiza un análisis de imágenes mediante una red neuronal para esto es necesario un Data set que no es más un conjunto de datos, que habitualmente están estructurados, como ejemplo podríamos decir que una tabla de una base de datos de sql sería un data set. Por lo general en las redes neuronales los data sets están compuestos por datos en función a características que se puedan analizar de una imagen. Data y Label donde data se refiere al conjunto de estos datos y Label a la etiqueta.

Para hablar de una Red neuronal es primordial definir y entender lo que es un Perceptrón. El modelo matemático más simple de una neurona es un perceptrón. La neurona es una célula especializada y caracterizada por poseer una cantidad indefinida de canales de entrada llamados dendritas y un canal de salida llamado axón.

En la que tenemos un conjunto de datos de entrada, una función de activación, y los resultados de salida.

Una red neuronal nace a partir del conjunto de perceptrón usados. Esto nos permite simular el comportamiento del cerebro humano. Una red neuronal artificial está formada por neuronas artificiales, que son unidades o nodos que reciben información del exterior o de otras neuronas, de manera similar a los impulsos nerviosos que reciben las neuronas del cerebro humano, las procesan y generan un valor de salida que alimenta a otras neuronas de la red o son la salida hacia el exterior de la red.

Como podemos observar al igual que el perceptron tenemos la primera columna como la entrada de los datos. Una capa oculta para el procesamiento de los datos, y una capa de salida para los resultados. La cantidad de capas de entrada y de salida las define el objetivo de la red neuronal. Mientras que el número de capas ocultas está dado por el número de procesamiento de datos que deseemos aplicar para realizar más precisa nuestra red.

# Librerías

Para el desarrollo de un clasificador de imágenes es necesario importar y conocer algunas librerías de Python que puedan facilitar el manejo de los datos una red neuronal.

Numpy es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.

Pytorch es un frameworks de redes neuronales, un conjunto de librerías y herramientas que nos hacen la vida más fácil a la hora de diseñar, entrenar y poner en producción nuestros modelos de Deep Learning. Una forma sencilla de entender qué es Pytorch es la siguiente:

Pytorch=Numpy + Autograd + GPU

De la misma manera que en Numpy el objeto principal es el ndarray, en Pytorch el objeto principal es el tensor. Podemos definir un tensor de manera similar a como definimos un arreglo, incluso podemos inicializar tensores a partir de arreglos.

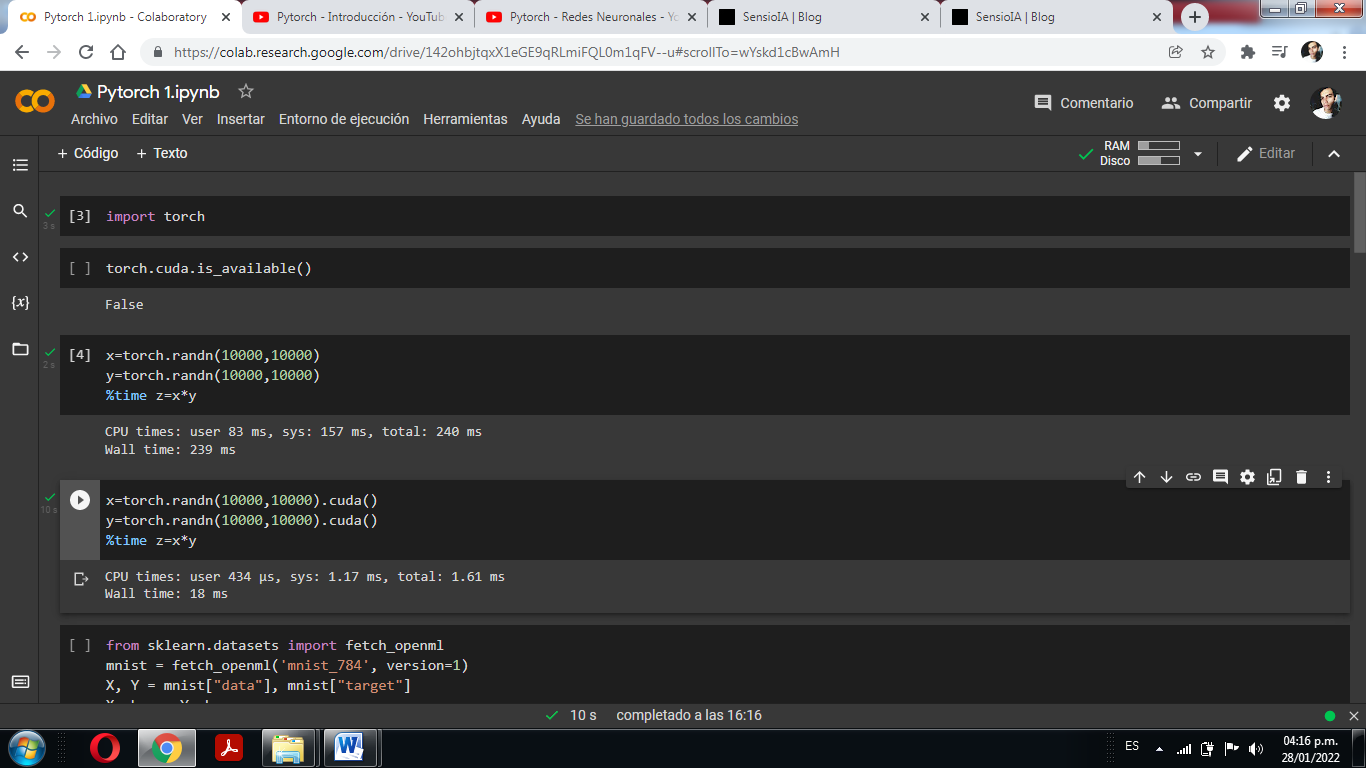
Scikit-learn es una biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python. ... Está diseñada para interoperar con las bibliotecas numéricas y científicas Numpy.

De esta manera es posible cargar data sets con datos reales para el uso de estas en los entrenamientos de las redes neuronales.

# Pytorch

¿Por qué usar pytorch?

Tensor flow. Rápidamente se convirtió en un “framework” de referencia de Deep Learning. Pero el problema principal por el que no nos decidimos a usar este framework era debido a la escasez de información científica que se encuentra actualmente en internet. Además que las soluciones para la clasificación de imágenes no es la más rápida. Mientras que por otro lado Pytorch nos ofrece la posibilidad de trabajar directamente en la GPU optimizando la velocidad enormemente.



Esta es la comparación en tiempo al realizar una tarea grande. Donde el tiempo de cada uno es de 239 ms de la CPU contra 18 ms en la GPU

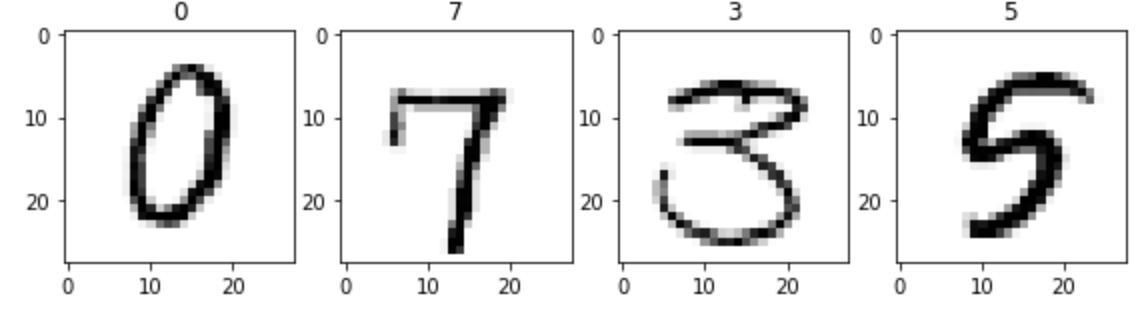
En resumen: Pytorch nos ofrece una mayor cantidad de información de carácter científico, nos ofrece la posibilidad de obtener derivadas casi de forma automática, un mayor control sobre las operaciones que estamos realizando y una mejor agilidad en los procesos.

# Código para el entrenamiento de la red neuronal

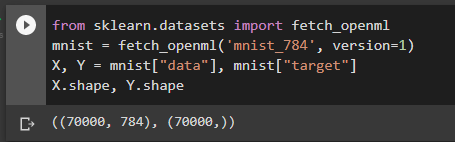
Para importar la librería de Pytorch usamos el siguiente comando.



Para este programa importamos el data set de MNIST que es una data set con 70,000 imágenes de 28x28 pixeles con dígitos escritos a mano como se muestra a continuación.



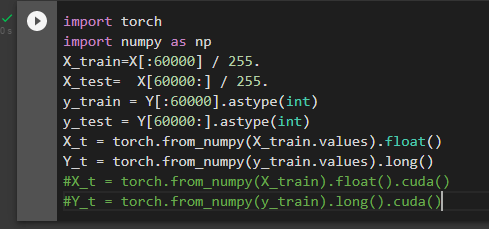
Para cargar esta date set usamos el siguiente código.



Donde en X guardamos el conjunto de datos. Mientras que en Target guardamos los labels a los que pertenece cada uno.

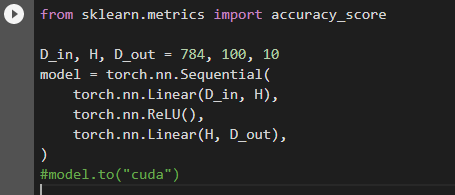
Ahora se debe separar los datos de entrenamiento con los datos de prueba, donde guardamos 60 mil datos para realizar el entrenamiento, y el resto para pruebas. Los normalizamos, los dividimos entre 255 para que trabajemos con números más pequeños y el proceso de entrenamiento sea más rápido. Después de esto convertimos los datos de arreglos de Numpy a tensores de Pytorch para poder trabajar con ellos.

Para esto importamos la librería Numpy y podemos nombrarla como np para facilitar el uso de esta.



La forma más sencilla de definir una red neuronal en Pytorch es utilizando la clase **Sequentail**. Esta clase nos permite definir una secuencia de capas, que se aplicarán de manera secuencial (las salidas de una capa serán la entrada de la siguiente). Es la forma ideal para definir un perceptron multicapas.

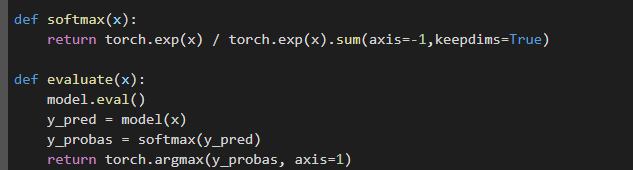
Para analizar una imagen dentro de una red neuronal es necesario descomponer en pixeles esta imagen. Por ejemplo. Si nuestra imagen es de 28x28. Nuestra red neuronal va a tener en total 784 entradas. Porque cada pixel de nuestra imagen va a representar un nodo de entrada de nuestra red neuronal. En este caso tenemos 10 salidas porque son 10 dígitos que pueden ser resultantes a la salida de la red neuronal.



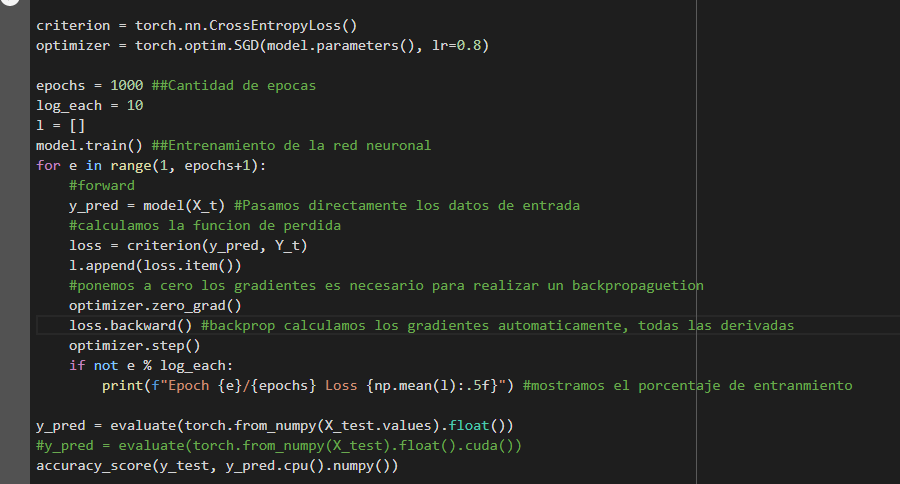
Tenemos 784 entradas conectadas en 100 neuronas, en la función Linear que nos permite plantear el modelo de la red neuronal, una función de activación ReLU, y por ultimo conectar nuestras 100 neuronas a las 10 salidas.

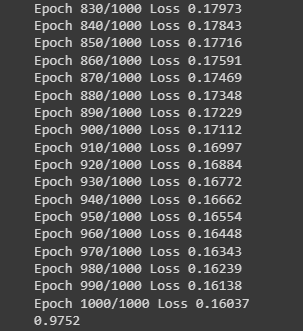
Función softmax es la más usada a la hora de clasificar objetos puesto que convierte cada uno de los valores resultantes de una transformación lineal realizadas en las neuronas a probabilidades que nos indican que tan probable es que determinada imagen pertenezca a un objeto en particular.

Model.Eval va a agarrar nuestro modelo y va asegurarse de que todas las capas estén listas para generar una predicción.



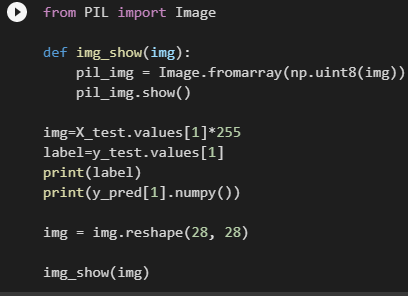
Llamamos a nuestra función de perdida crossentropy asignamos la cantidad de épocas al entrenamiento, donde directamente usamos nuestro modelo.

****

Lo aplicamos a mil épocas para obtener un porcentaje de perdidas más bajo, como podemos ver a las mil épocas tenemos un porcentaje de pérdida de 0.16037 lo cual es lo más acercado a 0 y está bien.

El resultado de la evaluación del modelo también nos da una precisión del 97.52% lo que quiere decir que puede tener un error en 2 de 100 intentos.

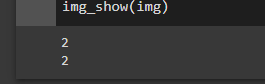
Con el siguiente código podemos realizar una prueba con uno de los datos de prueba y ver la precisión de la predicción.



Esta es la imagen de prueba y este el resultado que nos arrojó la predicción.



Donde el primero es el dato real y el segundo el dato que nos da el resultado de la clasificación.



# Conclusiones

Las redes neuronales son de gran ayuda a la hora de clasificar objetos, la creación de estos modelos sirve de gran ayuda para agilizar los procesos cuando se trata de obtener más resultados en las salidas. Es decir, clasificar uno o más objetos.

Actualmente me encuentro comenzando a entrar en estos temas de inteligencia artificial, machine learnig y sobre conocimiento de redes neuronales su funcionamiento más a fondo. Esta semana el siguiente reto es poder exportar el modelo ya entrenado y poder usarlo para clasificar dentro de otro entorno, conocer sobre redes neuronales convolucionales.