**¿Qué es el aprendizaje profundo?**

Subcategoría de ML que crea diferentes niveles de abstracción que representa los datos

**¿Qué es RELu?**

Es una función de activación, permite el paso de todos los valores positivos sin cambiarlos, pero asigna todos los valores negativos a 0

**TensorFlow**

Biblioteca de código abierto desarrollado por Google capaz de construir y entrenar redes neuronales.

Tensor Flow puede ser usado para ayudar al diagnóstico médico, detectar objetos, procesar imágenes, detección de emociones en el rostro, entre otras aplicaciones. En este curso usamos Tensor Flow para crear nuestra primera red neuronal y diseñar un clasificador de imágenes a partir de un conjunto de datos.

**Red neuronal Convolucional**

Modelan de forma consecutiva pequeñas piezas de información, al final combinan información en las capas más profuncas de la red

**Pasos para el código**

**Primero:** Importamos la biblioteca de tensorflow y las demás librerías

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

**Segundo:**

Cargamos el conjunto de datos de tensorflow

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

Además, creamos las clases

class\_names = [' T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneakers', 'Bag', 'Ankle boot']

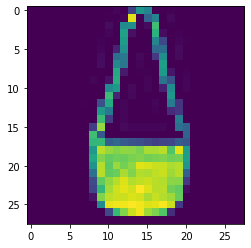
***Notita:*** train\_images.**shape** nos retorna el número de datos que tenemos.

(60000, 28, 28)

***Notita2:*** Si queremos mostrar alguna de las imágenes de nuestos datos podemos hacer lo sgte.

plt.figure()

plt.imshow(train\_images[520])



**Tercero:**

Procesamos las imágenes para que no sea complejo trabajar con ellos.

train\_images = train\_images/255

test\_images = test\_images/255

***Notita:*** Podemos mostrar y corroborar los cambios de esta manera

%matplotlib inline

plt.figure(figsize = (10, 10))

for i in range(25):

plt.subplot(5, 5, i + 1)

plt.xticks([])

plt.yticks([])

plt.grid('off')

plt.imshow(train\_images[i], cmap = plt.cm.binary)

plt.xlabel(class\_names[train\_labels[i]])



**Building the model**

The first layer in this network,[**tf.keras.layers.Flatten**](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten)**,** **transforms the format of the images** from a two-dimensional array(of 28 by 28 pixels) **to a one-dimensional array** (of 28 \* 28 = 784 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. **This layer has no parameters to learn; it only reformats the data.**

After the pixels are flattened, the network consists of a sequence of two [**tf.keras.layers.Dense**](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense) layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 128 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes.

**model = keras.Sequential([keras.layers.Flatten(input\_shape =(28, 28)), keras.layers.Dense(128, activation = tf.nn.relu), keras.layers.Dense(10, activation = tf.nn.softmax)])**

### **Compile the model**

Before the model is ready for training, it needs a few more settings. These are added during the model's compile step:

* **Loss function** —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
* **Optimizer** —This is how the model is updated based on the data it sees and its loss function.
* **Metrics** —Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the images that are correctly classified.

model.compile(optimizer = "adam", loss = 'sparse\_categorical\_crossentropy', metrics = ['Accuracy'])

**Páginas para datasets**

Kaggle.com

UCI machine learning Repository