# Autoencoder y clasificador convolucional

Ramiro Santamaria*
*Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires*
(Dated: February 5, 2024)

En este trabajo se entrenó una red neuronal autoencoder convolucional, con $N = 64$ neuronas en su capa oculta, para reproducir los datos de entrada de imágenes provenientes del set de datos de Fashion-MNIST. Con la red una vez entrenada se construyó un clasificador que incorporase el codificador, perteneciente al autoencoder, en su arquitectura. El clasificador tendrá como elementos de entrada las imágenes del set de datos y como salida la probabilidad que ésta pertenezca a alguna de las 10 etiquetas posibles.

## I. INTRODUCCIÓN

Los autoencoders son redes neuronales diseñadas para incorporar la esencia de datos complejos a través de un proceso de codificación y decodificación. Este enfoque se basa en la capacidad de la red para aprender representaciones eficientes y compactas de los datos de entrada. Su estructura se puede dividir en dos etapas principales, un codificador (encoder) y un decodificador (decoder). El encoder transforma la entrada original en una representación de dimensionalidad reducida, llamada "capa oculta" con longitud $N$. Esta capa busca capturar las características más importantes de los datos. Por otro lado el decoder intenta reconstruir la entrada original a partir de esta representación más compacta.


FIG. 1. Ejemplos del tipo de muestras del set de datos Fashion-MNIST.

En el siguiente trabajo se entrenó un autoencoder convolucional para reproducir el set de datos de Fashion-MNIST, que consta con 60000 datos de entrenamiento y 10000 datos de testeo. Cada unos de ellos es una imagen de $28x28$ píxeles en escala de grises y que pueden pertenecer a 10 clases diferentes [1]. Un ejemplo de las muestras utilizadas se presentan en la Fig. 1.

Una vez entrenado el autoencoder se utilizó la estructura de su encoder para construir un clasificador que tenga como elementos de entrada las imágenes y como conjunto de salida las 10 posibles etiquetas que representen las categorías posibles de los elementos de entrada.

## II. METODOLOGÍA

Como se mencionó en la sección anterior, en este trabajo se entrenó un autoencoder covolucional con el fin de replicar el set de datos de Fashion-MNIST. Con la red una vez entrenada se reutilizó la parte encoder de la misma para armar una red clasificadora que identifique las imágenes del set de datos según a la categoría a la cual pertenecen.

En las siguientes subsecciones se detallaran las arquitecturas de los modelos utilizadas para el autoencoder y para el clasificador.

### A. Autoencoder

Los autoencoder convolucionales se caracterizan por incorporar capas en su arquitectura que les ayuda a procesar datos con estructura espacial. Dichas capas aplican operaciones de convolución para detectar patrones de los datos de entrada, en nuestro caso imágenes.

El esquema del autoencoder utilizado en este trabajo se muestra en la Fig. 2, y se puede separa entre la parte encoder y decoder. La arquitectura del encoder se estructuró, de izquierda a derecha, de la siguiente manera [2]:

- Primero una capa *Convolucional* que le aplica distintos filtros a la imagen de entrada para detectar las diferentes características en paralelo. La aplicación de esta clase de capa produce una reducción en la dimensión de la imagen. Con lo cual se pierde cierta información en pos de detectar patrones de la misma.

- Luego se aplica una capa *Dropout* que se encarga de apagar conexiones entre capas contiguas con determinada probabilidad $p$. El objetivo es que se ayude a prevenir la dependencia entre las distintas conexiones entre las capas.

- La imagen luego es procesada por la función *MaxPooling* que reduce la dimensión espacial de

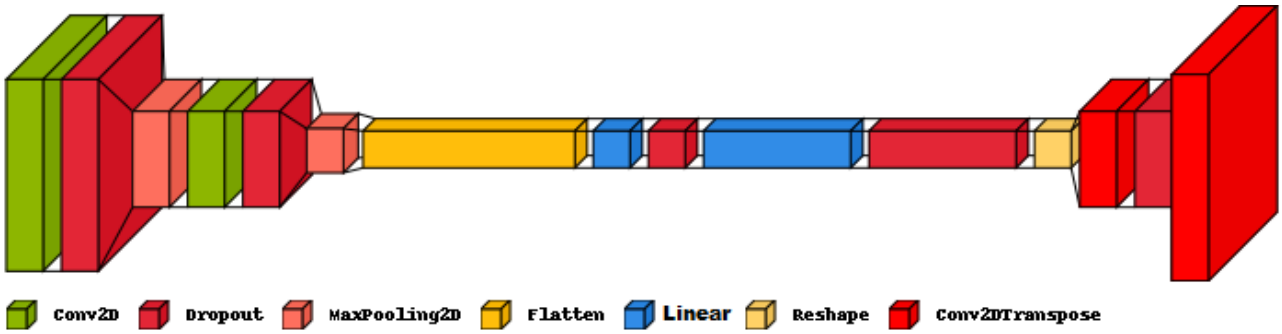


FIG. 2. Esquema de la arquitectura del autoencoder utilizado.

la misma, pero al mismo tiempo nos deja con la información más importante. Esto lo logra dividiendo los datos de entrada en diferentes regiones (ej. 2x2, 3x3, etc) y quedándose con el máximo valor de la región. Este proceso nos ayuda a que la red se quede con las características más importantes de los datos de entrada.

- Luego se aplica una nueva capa *Convolucional* seguida de un *Dropout* para proseguir con una nueva $MaxPooling$ y volver a redimensionar la imagen.
- A la estructura de los datos resultantes se le aplica la función *Flatten*, la cual lleva a los datos de tener estructura bidimensional a un vector con una sola dimensión (en nuestro caso de longitud 800).
- Para terminar la etapa de encoder, al vector resultante se le aplica una transformación lineal *Linear* para llevarlo a tener dimensión $N$ (64 en nuestro caso). Esta capa será la que contenga la representación comprimida y abstracta de la imagen.

Luego la tarea del decoder es a partir de los datos decodificados del encoder, realizar el camino inverso para reconstruir la imagen original. La arquitectura utilizada fue la siguiente:

- Una transformación *Linear*, para llevar el vector con longitud $N$ a uno con longitud 800. Para luego aplicar un *Dropout*.
- Para convertir el vector en un objeto bidimensional se le aplica la función $Unflaten$ (o *Reshape* en la Fig. 2).
- Por último realiza la operación $ConvTranspose2d$ casi de manera consecutiva, con un *Dropout* en medio, que nos devuelve nuevamente una imagen con dimensión igual a la de entrada.

Se tomo como funciones de activación entre todas las capas la función $ReLU(x)$, salvo en la salida del decoder en la que se tomó la función $Sigmoid(x)$.

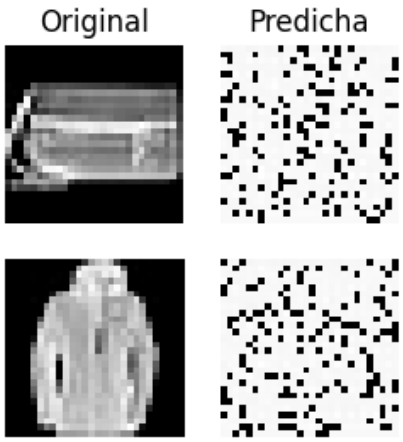


FIG. 3. Acción del modelo de autoencoder sin entrenar sobre dos muestras del set de datos.

A los elementos de salida del modelo se los compara con los valores esperables, que para un autoencoder son los datos de entrada, y se calcula que tan "lejos" están uno del otro. Para comparar el error de la predicción de la imagen formada por el modelo y la imagen original se utilizó el error cuadrático medio (ECM):

$$ECM = \frac{1}{M}\sum_{i=1}^{M}(Y_i - \hat{Y}_i)^2 \tag{1}$$

con $M$ el número de muestras, $\hat{Y}_i$ el valor obtenido por el modelo e $Y_i$ el valor esperado, que para el autoencoder es igual al dato de entrada (la imagen).

Luego de calcular los errores se computa el gradiente con respecto a todos los parámetros, y se utilizan para ajustar los pesos del modelo durante el entrenamiento.

En la Fig. 3 se muestra la acción del modelo sin entrenar, en donde vemos píxeles negros o blancos en su mayoría debido a la acción de la $Sigmoid(x)$.

Los detalles de la arquitectura del modelo se encuentran en el Apénd. A 1.

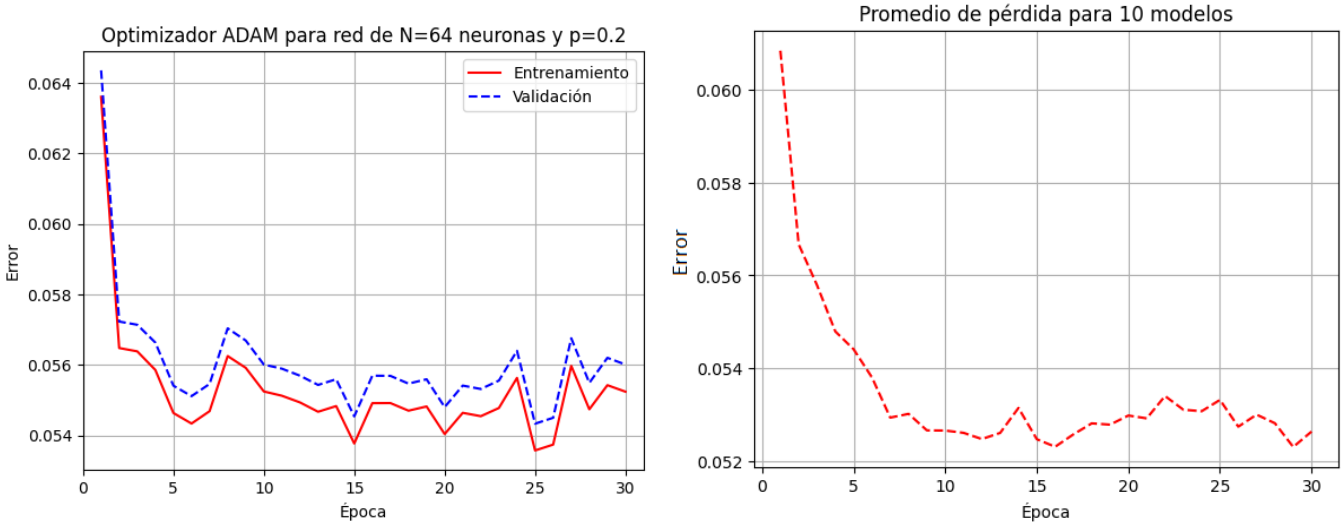


FIG. 4. En el panel izquierdo se muestra la evolución de los errores, de entrenamiento en rojo y validación en azul, del autoencoder a lo largo de las épocas. En el panel de la derecha vemos el promedio de diez corridas de entrenamiento del modelo.

### B. Clasificador

El clasificador que se utilizó en este trabajo consta de dos estructuras principales. La primera será la de utilizar la parte encoder del autoencoder entrenado, que nos dará una representación comprimida de la imagen en un vector de longitud $N$. Y la segunda será una capa lineal que nos lleve de un vector de longitud $N$ a uno de longitud 10, que son las posibles etiquetas que pueden tener nuestro set de datos de imágenes.

Para entrenar al clasificador es necesario definir una función error para determinar que tan bien lo está haciendo nuestro modelo y en que medida se deberán ajustar los pesos del modelo. Dado que el problema es de clasificación se suele utilizar la función de pérdida la entropía cruzada:

$$H(y, \hat{y}) = -\sum_i y_i log(\hat{y_i}) \tag{2}$$

siendo $\hat{y_i}$ la probabilidad predicha por el modelo para la clase $i$, y $y_i$ es la etiqueta verdadera para la clase $i$.

A partir de estos errores se computa el gradiente con respecto a todos los parámetros, y se utilizan para ajustar los pesos del modelo durante la etapa de entrenamiento.

La estructura de código utilizada para el clasificador se muestra en el Apénd. A 2.

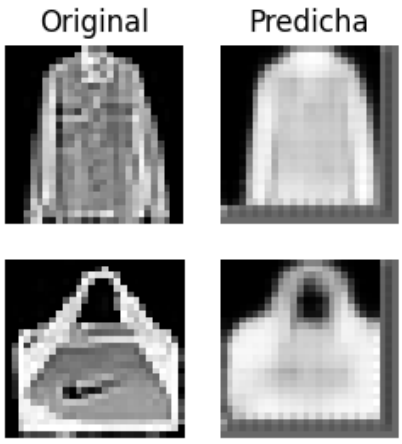


FIG. 5. Ejemplos de la acción del autoencoder entrenado sobre dos muestras del set de datos.

## III. RESULTADOS Y DISCUSIONES

En las siguientes subsecciones se detallarán los hiperparámetros y criterios utilizados para el entrenamiento de los dos modelos, el autoencoder y el clasificador. En ambos casos se dividió los 60000 datos de entrenamiento en dos submuestras, 500000 que se usarán para entrenar los modelos y 10000 para su validación. Esta división nos permite identificar si nuestros modelos están sobreajustando las muestras de entrenamiento y su dificultad que tengan para generalizar los datos. Por último se tomaron los 10000 datos de testeo y se evaluó su error.

Si no se aclara lo contrario se utilizó $N = 64$ para el número de neuronas en la capa oculta, la última del encoder, y un $Dropout = 0.2$ para ambos modelos.

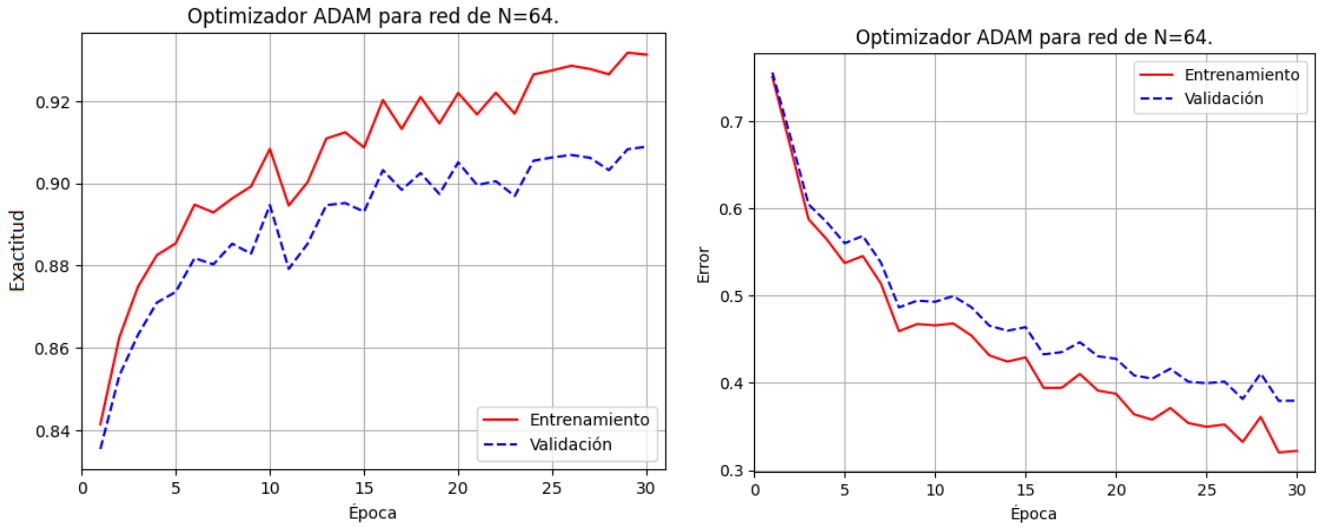


FIG. 6. Evolución de la exactitud (panel izquierdo) y del error (panel derecho) para los datos de entrenamiento y validación del clasificador.

### A. Autoencoder

En el panel izquierdo de la Fig. 4 vemos un ejemplo de aprendizaje del autoencoder. La curva roja continua muestra la evolución del error de entrenamiento a lo largo de las épocas, y la línea punteada azul la evolución de los errores de validación. Dado la forma "picuda" de las curvas se entrenaron 10 modelos durante las mismas épocas y se realizó un promedio para poder analizar más en claro el comportamiento de la evolución del error de entrenamiento. El resultado se muestra en el panel derecho de la Fig. 4. Podemos notar que en promedio el error es decreciente desde la época 0 hasta la época 10, y luego comienza a oscilar alrededor de algún valor. Con lo cual se decidió entrenar al modelo final hasta la época 15, ya que más iteraciones no brindaría mejoras al modelo.

Como error de testeo para el modelo de autoencoder se obtuvo 0.0533, y la acción del mismo sobre las imágenes del set de datos se muestran en en la Fig. 5. Vemos que hay una buena reproducción en los contornos de las prendas, así también la pérdida de píxeles en el lado derecho e inferior de las imágenes debido a la acción de las capas convolucionales.

### B. Clasificador

Una vez entrenado el autoencoder se utilizó la parte encoder del mismo para nuestro clasificador y se prosiguió al entrenamiento del mismo.

Además de la evolución del error a lo largo del entrenamiento del modelo, según la Ec. 2, es necesario cuantificar que tan bien está clasificando las imágenes de entrada con sus respectivas etiquetas. Como medida del desempeño del clasificador se tomó la *Exactitud* [3]:

$$Excatitud = \frac{\#de\ predicciones\ correctas}{total\ de\ \#\ de\ predicciones} \qquad (3)$$

que nos indica el número total de predicciones correctas pesado sobre el número total de predicciones.

En la Fig. 6 vemos la evolución de la *Exactitud*, panel izquierdo, y del error, panel derecho, a lo largo del período de entrenamiento. Podemos notar tanto las curvas de validación como la de entrenamiento siempre mantienen las mismas tendencias, y que el modelo siempre tiene un mejor desempeño con los datos de entrenamiento (que es lo esperable). Además se observó que para entrenamientos mayores a 20 épocas los modelos no mostraban mejoras sustanciales en su desempeño. Con lo cual se decidió entrenar el modelo hasta la época 30.

El error del clasificador final para $N = 64$ sobre los datos de testeo arrojó un valor de 0.3865, y una $Exactitud = 90,43\%$.

Una manera de cuantificar el rendimiento de nuestro modelo es a través de la matriz de confusión, en donde el número de predicciones correctas e incorrectas se resumen con los valores de conteo y se desglosan por cada clase [3].

En la Fig. 7 podemos ver el desempeño de nuestro clasificador para todas las etiquetas posibles del set de datos sobre los datos de testeo. Sobre el eje vertical se grafican las etiquetas verdaderas y sobre el eje horizontal las predicciones del modelo. Los elementos de la diagonal representan el porcentaje de etiquetas que el modelo clasificó exitosamente, y los elementos fuera de la diagonal el porcentaje en el cual el modelo no tuvo éxito en la clasificación [4]

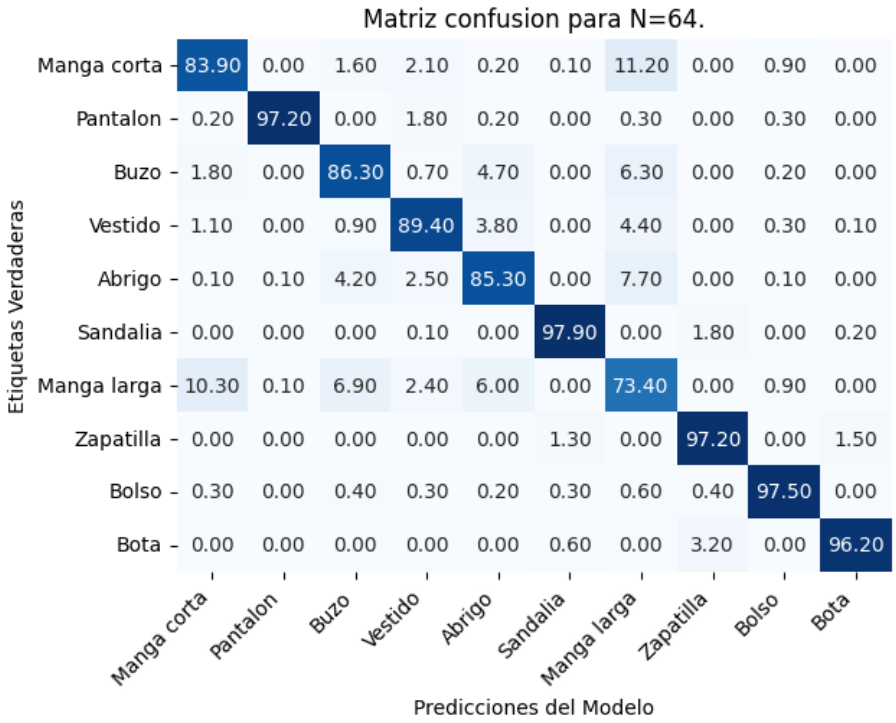


FIG. 7. Matriz de confusión para el clasificador ya entrenado, con $N = 64$ neuronas en su capa oculta.

Podemos observa que en general nuestro modelo clasifica con un porcentaje mayor al 90%, salvo a los elementos que son similares. Vemos que el modelo clasifica las imágenes con *Manga larga* en un porcentaje no despreciable con elementos que comparten patrones similares, como pueden ser *Manga corta*, *Buzo* o *Abrigo*.

Además de estudiar el clasificador con $N = 64$ neuronas en su capa oculta también se analizaron los casos para $N = 128$ y $N = 256$, pero no se encontraron mejoras significativas respecto al modelo presentado en este trabajo. En particular el clasificador con $N = 128$ arrojó una $Exactitud = 91.41\%$, y para el caso $N = 256$ una $Exactitud = 90.7\%$.

## IV. CONCLUSIONES

En este trabajo se entrenó un autoencoder convolucional con $N = 64$ neuronas en la capa oculta para reconstruir las muestras de entrada del set de datos de Fashion-MNIST. Consta de 60000 datos de entrenamiento y de 10000 datos de testeo que se utilizaron para calcular el error final del modelo.

Para cuantificar las diferencias entre las imágenes esperadas y las predicciones que realizó el modelo de autoencoder se utilizó el error cuadrático medio (ECM). El error final para el modelo resultó de 0.0533 para los datos de testeo.

Con el autoencoder ya entrenado, se tomó la parte encoder del mismo para reutilizarlo en un modelo de clasificación. El objetivo es que el clasificador tome de entrada las imágenes del set de datos y nos devuelva la probabilidad que tenga la imagen de pertenecer a una de las 10 etiquetas posibles que tiene el set de datos.

El clasificador se entrenó tomando como a la función error la entropía cruzada. Que como error de testeo final al modelo de clasificador utilizado arrojó 0.3865, y que obtuvo un valor de $Exactitud = 90.4\%$.

Mediante la matriz de confusión de las etiquetas verdaderas y las predicciones del modelo se observó un buen porcentaje de predicciones correctas, segmentado por las diferentes etiquetas. Encontrando que el modelo se lo dificulta más distinguir las imágenes que compartieran características importantes, como puede ser un remera de manga larga con un buzo, abrigo o una remera de manga corta.

* rami.santamaria92@gmail.com

[1] Fashion mnist.
[2] Pytorch.
[3] Comprensión de la matriz de confusión y cómo implementarla en python (2020).
[4] Matriz de confusión.

## Appendix A: Modelos

### 1. Autoencoder

```
class Autoencoder(nn.Module):
    def __init__(self,n,p):
        super(Autoencoder,self).__init__()
        self.flatten = nn.Flatten()
        self.encoder = nn.Sequential(
            # Capa convolucional 2D compuesta
            nn.Conv2d(1,16,kernel_size=3),
            #  Dim:(1,28,28) --> (16,26,26)
            nn.ReLU(),
            nn.Dropout(p),
            nn.MaxPool2d(2,2),
            # Dim:(16,26,26) --> (16,13,13). Crea con un tamaño de kernel de 2x2 y un paso de 2.
            nn.Conv2d(16,32,kernel_size=3),
            # Dim:(16,13,13) --> (32,11,11)
            nn.ReLU(),
            nn.Dropout(p),
            nn.MaxPool2d(2,2),
            # Dim:(32,11,11) --> (32,5,5)
            nn.Flatten(),
            # Dim:(32,5,5) --> 32*5*5
            nn.Linear(32*5*5,n),
            # Dim:32*5*5 --> n
            nn.ReLU(),
            nn.Dropout(p),
         )
        self.decoder = nn.Sequential(
            # Capa lineal
            nn.Linear(n,32*5*5),
            nn.ReLU(),
            nn.Dropout(p),
            nn.Unflatten(1,(32,5,5)),
            # Dim:32*5*5 --> (32,5,5)
            nn.ConvTranspose2d(32,16,kernel_size=(4,4),stride=(2,2),output_padding=(1,1)),
            # Dim:(32,5,5) --> (16,13,13)
            nn.ReLU(),
            nn.Dropout(p),
            nn.ConvTranspose2d(16,1,kernel_size=(3,3),stride=(2,2),output_padding=(1,1)),
            # Dim:(16,13,13) --> (1,28,28)
            nn.Sigmoid(),
            nn.Dropout(p),
        )
    def forward(self,x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

### 2. Clasificador

```
    class Clasificador(nn.Module):
     def __init__(self,encoder):
         super(Clasificador,self).__init__()
```

```
        self.flatten = nn.Flatten()
        self.encoder = encoder
        self.cl      = nn.Sequential(
            nn.Linear(64,10),
            nn.ReLU(),
            nn.Dropout(0.2),
        )
    def forward(self,x):
        x = self.encoder(x)
        x = self.cl(x)
        return x
```