UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

SEMINARIO DE TESIS I:

Uso de técnicas de desenfoque con Redes Neuronales Convolucionales

ELABORADO POR:

Rommel Yoshimar Condori Muñoz

ASESOR:

Yuri Javier Ccoicca Pacasi

(Lima - Perú)

2023

Índice general

Índice de figuras

Resumen

El desenfoque de imágenes es un problema clásico en la visión por computadora de bajo nivel con el objetivo de recuperar una imagen nítida de una imagen de entrada borrosa. Los avances en el aprendizaje profundo han llevado a un progreso significativo en la solución de este problema y se han propuesto una gran cantidad de redes de desenfoque. Este documento presenta una solución oportuna de desenfoque de imágenes basados en el aprendizaje profundo. A continuación, se presenta un método que utiliza las redes neuronales convolucionales (CNN) basada en arquitectura, función de pérdida y aplicación. Después,se tuvo que desenfocar un conjunto de datos de imagenes conocida proporcionada por keras(Fashion-Mnist) y con éstas entrenar el modelo para finalmente predecir el resultado de las imágenes modificadas. Concluimos que el conjunto de imágenes predichas estan muy cercanos a parecerse al original.

Capítulo 1

Introducción

La presencia de imágenes borrosas es uno de los desafíos más frecuentes en la fotografía. Aunque pueden parecer nítidas en un primer momento, al transferirlas al ordenador, la falta de nitidez las convierte en instantáneas inservibles.

Para abordar este caso, me apoyaré en las redes neuronales convolucionales, un tipo de red neuronal especializada en el procesamiento de imágenes. Estas redes tienen la capacidad de extraer características ocultas de las imágenes, que pueden ser imperceptibles al ojo humano. Debido a estas características, las redes neuronales convolucionales son ampliamente utilizadas en diversas aplicaciones de visión artificial, como reconocimiento, localización y detección de objetos, entre otras.

Es importante saber qué tan útiles son en esta aplicación de desenfocar una imagen. Los codificadores automáticos usan estas Redes Neuronales Convolucionales, de manera óptima para resolver este problema.

1.1. Objetivos

1.1.1. Objetivo General

Lograr desenfocar imágenes implicando conceptos de Redes Neuronales Convolucionales.

1.1.2. Objetivo Específico

- Difuminar imágenes.
- Construir una arquitectura de red neuronal convolucional.
- Hacer que el modelo se ajuste a los datos de entrada.
- Predecir con el modelo entrenado para desenfocar la imagen.

Capítulo 2

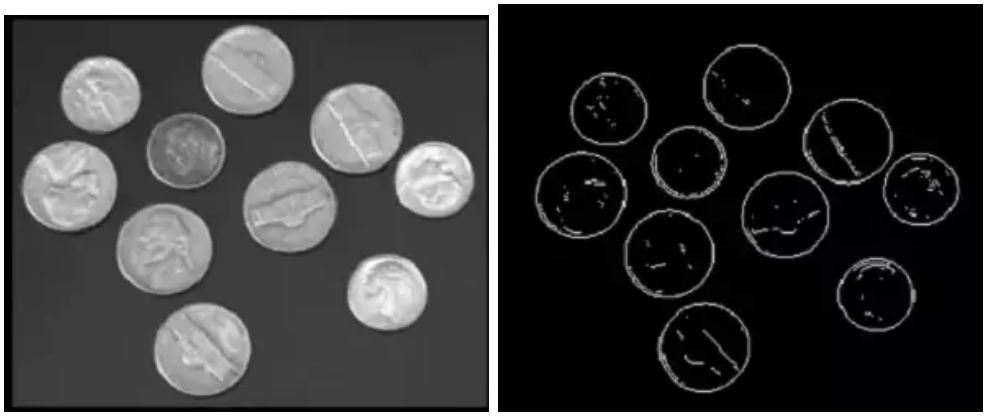
Estado del Arte

2.1. Conceptos previos

2.1.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales [1] es un algoritmo de Deep Learning que está diseñado para trabajar con imágenes, tomando estas como input, asignándole importancias (pesos) a ciertos elementos en la imagen para así poder diferenciar unos de otros. Este es uno de los principales algoritmos que ha contribuido en el desarrollo y perfeccionamiento del campo de Visión por computadora. Las redes convolucionales contienen varias hidden layers, donde las primeras puedan detectar líneas, curvas y así se van especializando hasta poder reconocer formas complejas como un rostro, siluetas, etc. Las tareas comunes de este tipo de redes son: Detección o categorización de objetos, clasificación de escenas y clasificación de imágenes en general. La red toma como entrada los pixeles de una imagen.

- Kernel: El kernel en las redes convolucionales se considera como el filtro que se aplica a una imagen para extraer ciertas características importantes o patrones de esta.



(a) Antes (b) Después

Figura 2.1: Aplicando el filtro o kernel.

- Convolución: Uno de los procesos más distintivos de estas redes son las convoluciones. El cual consiste en tomar un grupo de píxeles de la imagen de entrada e ir realizando un producto escalar con un kernel. El kernel recorrerá todas las neuronas de entrada y obtendremos una nueva matriz, la cual será una de las hidden layers. En el caso de que la imagen sea de color se tendrán 3 kernels del mismo tamaño que se sumarán para obtener una imagen de salida.

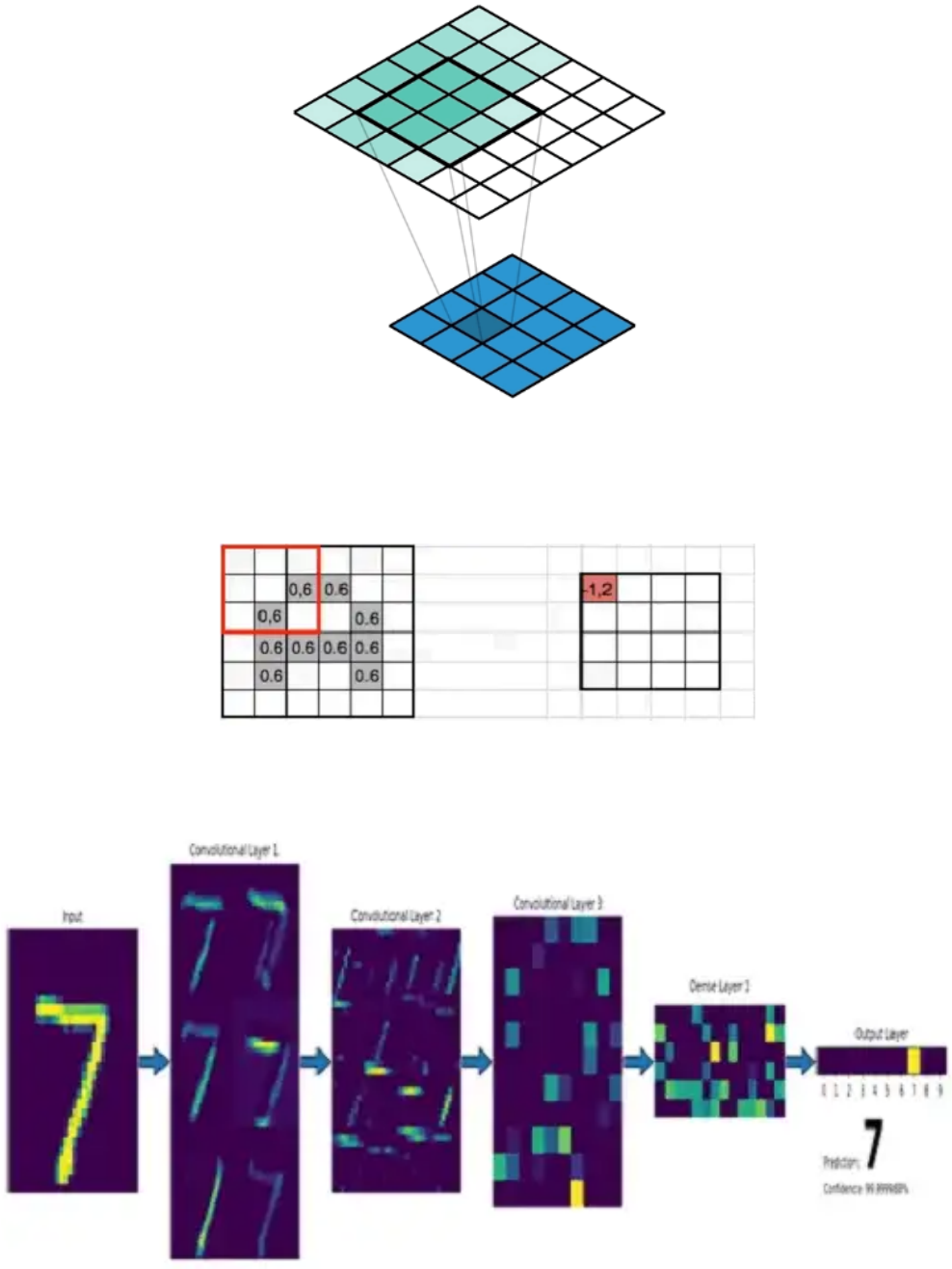


Figura 2.2: Proceso de convolucion.

2.1.2. Codificadores Automáticos

Los codificadores automáticos [5] se utilizan para tomar una imagen de entrada y luego almacenar los detalles de esa imagen en un formato diferente cuyo tamaño es más pequeño que el tamaño de la imagen. Estos detalles almacenados se pueden usar más tarde para recrear la misma imagen o una imagen diferente basada en la imagen de entrada. Los codificadores automáticos son el concepto fundamental detrás de la recreación de imágenes, además de generar nuevas imágenes y, por lo tanto, también se utilizan en muchas redes adversarias generales.

Un Auto-Codificador, en su conjunto, tiene 3 secciones, un codificador , una capa oculta y un decodificador . El codificador toma la entrada, la procesa, extrae características y luego almacena los datos en la capa oculta . Un decodificador hace lo contrario de lo que hace un codificador. Toma los datos de la capa oculta y luego recrea una imagen usando la misma.

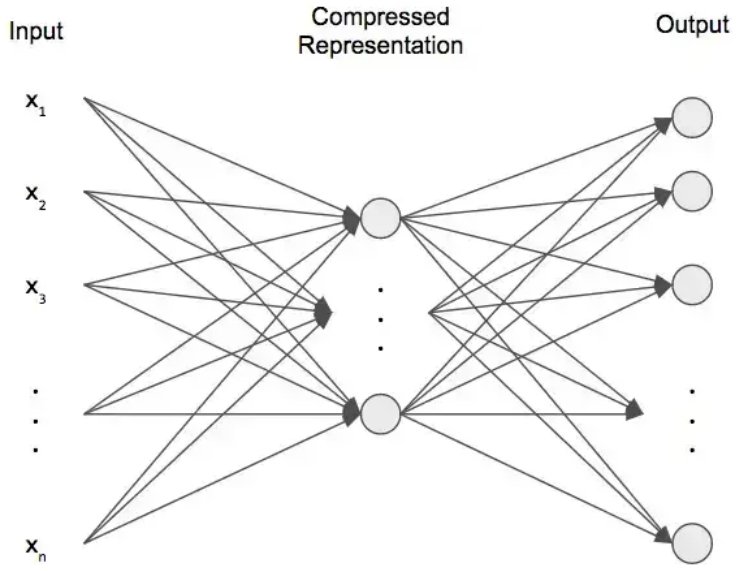


Figura 2.3: Un esquema básico de un codificador automático.

2.1.3. Funciones de pérdida

Una función de pérdida [5], o Loss function, es una función que evalúa la discrepancia entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje. Cuanto menor es el resultado de esta función, más eficiente es la red neuronal en su rendimiento. La minimización de esta función implica reducir al mínimo la discrepancias entre el valor predicho y el valor real para una observación dada, lograndose mediante el ajuste de los distintos pesos de la red neuronal.

- Error Cuadrático Medio: se define (como su nombre ya sugiere) como la media de la diferencia cuadrática entre la salida de nuestra red y la realidad del terreno. Cuando la salida del codificador es una cuadrícula de valores, también conocida como una

imagen , el MSE entre la imagen de salida I^- y la imagen real I puede definirse como

$$MSE = \frac{1}{NM}\sum_{i}^{N}\sum_{j}^{M}(\bar{I}_{ij} - I_{ij})^2$$

- Entropía Cruzada Binaria: Es la suma de las entropías cruzadas de píxeles entre la imagen predicha y la realidad del terreno. Está representado por:

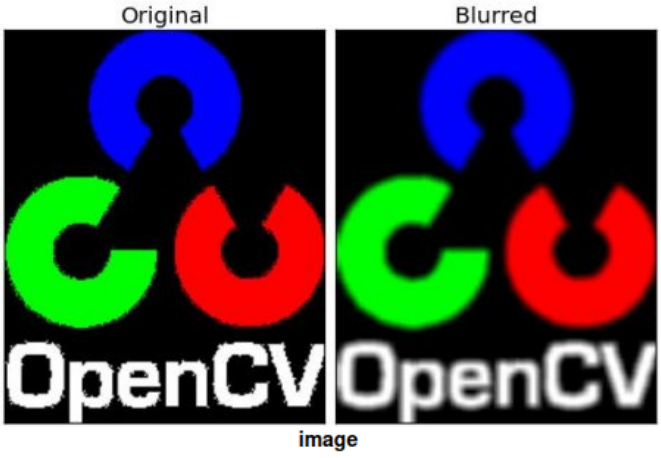
$$BCE = \sum_{i}^{N}\sum_{j}^{M}(\bar{I}_{ij}\log(I_{ij}) - I_{ij}\log(\bar{I}_{ij}))$$

2.1.4. Desenfoque de imágenes

El desenfoque de una imágenes [4] es una tarea clásica en la visión por computadora de bajo nivel, que ha atraído la atención de la comunidad de procesamiento de imágenes y visión por computadora. El objetivo del desenfoque de imagen es recuperar una imagen nítida de una imagen de entrada borrosa, donde el desenfoque puede ser causado por varios factores, como la falta de enfoque, el movimiento de la cámara o el movimiento rápido del objetivo.

El desenfoque de una imagen [2] se logra mediante la convolución de la imagen con un núcleo de filtro de paso bajo. Esta técnica es útil para eliminar el ruido presente en la imagen reduciendo así su contenido de alta frecuencia, que incluye elementos como el ruido y los bordes. Como resultado de esta operación, los bordes de la imagen se ven ligeramente desenfocados (aunque también existen técnicas de desenfoque preservan los bordes). La librería OpenCV proporciona cuatro técnicas principales de técnicas de desenfoque:

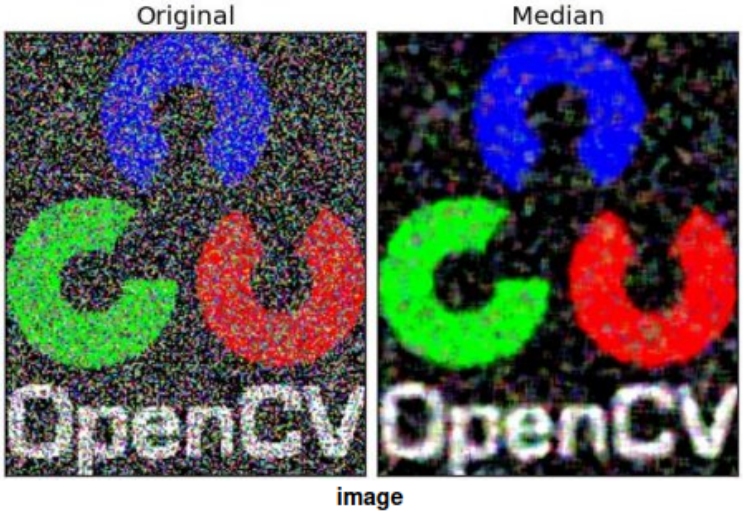
- Promedio : El desenfoque promedio se logra convolucionando una imagen con un filtro de cuadro normalizado. Esta técnica simplemente toma el promedio de todos los píxeles dentro del área del kernel y reemplaza el elemento central con el valor promedio resultante. La función çv.blur().º çv.boxFilter().[en] la librería OpenCV se utilizan para aplicar este tipo de desenfoque. Para ello, es necesario especificar el ancho y la altura del núcleo que define el tamaño del filtro de cuadro.



image

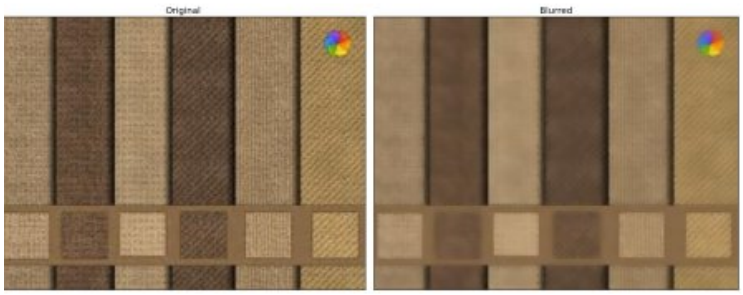
- Medio : Aquí, la función cv.medianBlur() toma la mediana de todos los píxeles debajo del área del kernel y el elemento central con este valor de la mediana . Ésta es especialmente efectiva contra el ruido de sal y pimienta presente en una imagen. A diferencia de otros filtros anteriores, donde el elemento central siempre se sustituye por un valor recién calculado o un valor completamente nuevo, en el desenfoque medio, el elemento central siempre se sustituye por un valor de píxel existente en la imagen. Es importante destacar que el tamaño del núcleo utilizado para aplicar este desenfoque debe ser un número entero impar y positivo.

  En esta demostración, se agrega un 50 % de ruido a la imagen original y aplica un desenfoque medio mediante la función çv.MedianBlur".



- Filtrado Bilateral : cv.bilateralFilter() es altamente efectivo en la eliminación de ruido mientras mantiene los bordes nítidos. Pero el funcionamiento es más lento en comparación con otros filtros. Ya vimos que un filtro gaussiano toma la vecindad alrededor del píxel y encuentra su promedio ponderado gaussiano. Este filtro gaussiano es una función del espacio únicamente, es decir, los píxeles cercanos se consideran durante el filtrado. No considera si los píxeles tienen casi la misma intensidad. No considera si un píxel es un píxel de borde o no. Entonces también difumina los bordes, lo cual no queremos hacer.

  El filtrado bilateral también toma un filtro gaussiano en el espacio, pero un filtro gaussiano más que es una función de la diferencia de píxeles. La función gaussiana del espacio asegura que solo los píxeles cercanos se consideren para el desenfoque, mientras que la función gaussiana de diferencia de intensidad asegura que solo aquellos píxeles con intensidades similares al píxel central se consideren para el desenfoque. Por lo tanto, conserva los bordes, ya que los píxeles en los bordes tendrán una gran variación de intensidad.



image

- Gaussiano : En este método, en lugar de utilizar un filtro de caja, se aplica un núcleo gaussiano mediante la función cv.GaussianBlur(). Para ello, debemos especificar el ancho y la altura del núcleo, que deben ser positivos e impares. Además, es necesario indicar la desviación estándar en las direcciones X e Y, sigmaX y sigmaY respectivamente. Si solo se especifica sigmaX, sigmaY se considera igual a sigmaX. En caso de proporcionar ambos valores como cero, se calcularán a partir del tamaño del kernel.

  El desenfoque gaussiano resulta altamente efectivo para eliminar el ruido gaussiano presente en una imagen.

image

Capítulo 3

Marco Metodológico

En este proyecto se construye una red neuronal en python para hacer un modelo de desenfoque de imágenes, optimizado por un algoritmo , para un conjuntos de datos que está incluido en el paquete Keras.

3.1. Herramientas

3.1.1. Software

Se utilizó principalmente:

- Python 3.9.13 (Anaconda3)
- Jupyter Notebooks (Google Collab)

3.1.2. Librerias

- Keras
- Tensorflow
- Pandas
- OpenCV
- Matplotlib.Pytplot
- h5py

3.1.3. Recursos

Conjunto de datos Fashion-MNIST [3]: Incluye una gran cantidad de variedades de imágenes proporcionada por la libreria Keras.

Fashion-MNIST es un conjunto de datos de imágenes de artículos de Zalando, que consta de un conjunto de entrenamiento con 60,000 ejemplos y un conjunto de prueba con 10,000 ejemplos. Cada ejemplo es una imagen en escala de grises de 28x28 píxeles, asociada con una etiqueta de una de las 10 clases posibles. La intención de Zalando al crear Fashion-MNIST es que este conjunto de datos funcione como un reemplazo directo del conjunto de datos MNIST original, permitiendo comparar algoritmos de aprendizaje automático. Fashion-MNIST comparte el mismo tamaño de imagen y la misma estructura de divisiones entre entrenamiento y prueba que el conjunto de datos MNIST original.

Cada imagen tiene 28 píxeles de alto y 28 píxeles de ancho, para un total de 784 píxeles en total. Cada píxel tiene un único valor de píxel asociado, que indica la claridad u oscuridad de ese píxel, donde los números más altos significan más oscuro. Este valor de píxel es un número entero entre 0 y 255. Los conjuntos de datos de entrenamiento y prueba tienen 785 columnas. La primera columna consta de las etiquetas de clase y representa la prenda de vestir. El resto de las columnas contienen los valores de píxel de la imagen asociada.

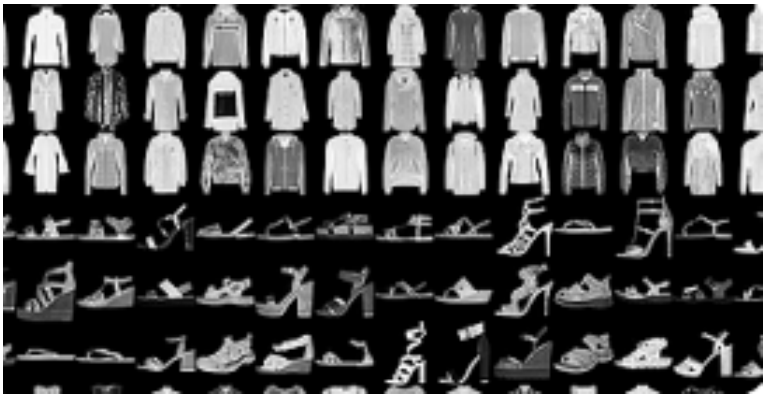


Figura 3.1: Conjunto de datos Fashion-MNIST

3.2. Metodología

3.2.1. Implementación de la Red Neuronal

Para la implementación del codificador, este solo va a consistir en varias capas convolucionales de un número diferente de filtros.

Estas capas extraerán características de la imagen de entrada, que será una imagen borrosa, y luego transfiere estas características a una capa oculta. Esta consta de una capa de convolución y la entrada a esta capa es el mapa de características o la salida del codificador.

Los datos de la capa oculta se pasan al decodificador . Dado que el codificador involucra capas de convolución, tiene sentido desconvolucionar para obtener una imagen similar a la imagen de entrada. En este caso, es la imagen desenfocada. En la deconvolución de imágenes, se tomará un núcleo con pesos, similar a una capa de convolución, y se multiplica por la intensidad de un solo píxel del mapa de características. Esta nueva

matriz reemplaza el píxel en el mapa de características. Los pesos de los núcleos de cada capa se aprenden durante el proceso de entrenamiento del modelo general.

Por lo tanto, el decodificador, involucra solo la capa de deconvolución o, a veces, incluso llamada capa de transposición de convolución . Las capas de desconvolución en el decodificador tienen parámetros y atributos similares a los elegidos para el codificador.

3.2.2. Método de aprendizaje

Se tiene la lista la arquitectura del autoencoder. Pero para entrenar el modelo, también se deberá elegir una función de pérdida . Hay bastantes funciones de pérdida disponibles para lograr el objetivo. Pero, en esta caso me enfocaré en el error cuadrático medio y la función de pérdida de entropía cruzada binaria .

Ambas funciones de pérdida nos dan una idea aproximada de la diferencia entre la realidad fundamental y la imagen predicha. Minimizar ambas pérdidas ayudará a resolver el ajuste de los pesos y sesgos utilizados en la CNN diseñada anteriormente.

3.2.3. Implementación del Codigo

El código se implementa utilizando el paquete Keras disponible en Python.

Cargo el conjunto de datos.

```
#carga del dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train, X_test = X_train/255, X_test/255
```

Como ninguna de las imágenes en el conjunto de datos está borrosa, creo un nuevo conjunto de datos a partir de las imágenes existentes haciendolas desenfocar. Para difuminarlos, uso la función GaussianBlur disponible en el paquete OpenCV . El tamaño del núcleo elegido será 3 x 3.

```
#aplicando el desenfocado
def add_noise(X):
    res = []
    for pic in X:
        blur = cv2.GaussianBlur(pic, (3, 3), 0)
        blur = np.clip(blur, 0, 1)
        res.append(blur)
    return np.array(res)
noise_train = add_noise(X_train)
noise_test = add_noise(X_test)
```

Ahora, construyo una arquitectura:

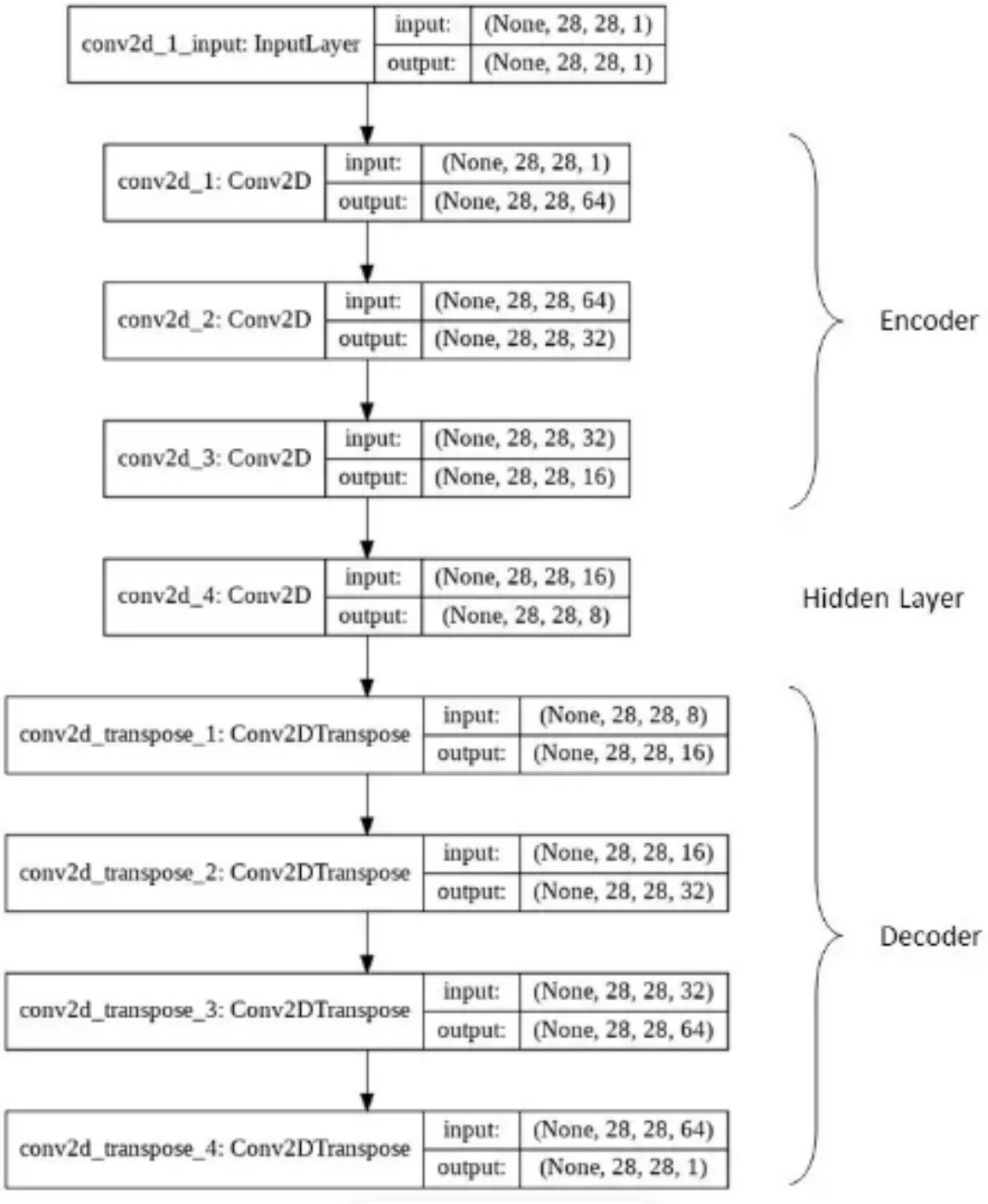


Figura 3.2: Arquitectura

Las primeras conv2d_1, conv2d_2, conv2d_3 representan el codificador , conv2d_4 representan la capa oculta y todas las demás capas Conv2DTranspose representan el decodificador . Tomando en cuenta que, la dimensión de salida debe ser la misma que la

dimensión de entrada. Esta arquitectura se puede codificar de la siguiente manera:

```
#creando el modelo

model = models.Sequential()

model.add(Conv2D(64, (2, 2), strides = 1, padding = 'same', input_shape = (28, 28, 1)))
model.add(Conv2D(32, (2, 2), strides = 1, padding = 'same'))
model.add(Conv2D(16, (2, 2), strides = 1, padding = 'same'))

model.add(Conv2D(8, (2, 2), strides = 1, padding = 'same'))

model.add(Conv2DTranspose(16, (2, 2), strides = 1, padding = 'same'))
model.add(Conv2DTranspose(32, (2, 2), strides = 1, padding = 'same'))
model.add(Conv2DTranspose(64, (2, 2), strides = 1, padding = 'same'))
model.add(Conv2DTranspose(1, (1, 1), strides = 1, activation = 'sigmoid', padding = 'same'))
```

Se puede tomar cualquiera de las funciones de pérdida, como se explicó anteriormente. Personalmente encontré mejores resultados con el error cuadrático medio para este conjunto de datos. Ahora, necesitamos compilar este modelo y luego ajustar los datos.

```
#entrenando el modelo
model.compile(loss = 'mse', optimizer = 'adam')
model.fit(noise_train.reshape(-1, 28, 28, 1),
          X_train.reshape(-1, 28, 28, 1),
          epochs = 100,
          batch_size = 2000,
          validation_data = (noise_test.reshape(-1  , 28, 28, 1),X_test.reshape(-1, 28, 28, 1)))
```

Una vez que el modelo se ajusta a los datos de entrada, ahora es el momento de predecir las imágenes desenfocadas.

Ahora bien, lo que se aplico hasta ahora fue con un desenfoque Gaussiano que me permite la libreria openCV, similarmente aplico para los tres restantes tipo de técnicas de desenfoque (Promedio, Medio y Filtrado Lateral) bajo el mismo modelo propuesto.

- Promedio

```
#aplicando el desenfocado promedio
def add_noise(X):
    res = []
    for pic in X:
        blur = cv2.blur(pic, (3, 3))
        blur = np.clip(blur, 0, 1)
        res.append(blur)
    return np.array(res)
noise_train = add_noise(X_train)
noise_test = add_noise(X_test)
```

- Medio

```
#aplicando el desenfocado medio
def add_noise(X):
    res = []
    for pic in X:
        blur = cv2.medianblur((pic*255).astype(np.uint8), 5
        blur = np.clip(blur, 0, 1)
        res.append(blur)
    return np.array(res)
noise_train = add_noise(X_train)
noise_test = add_noise(X_test)
```

- Filtrado Bilateral

```
#aplicando el desenfocado de filtrado bilateral
def add_noise(X):
    res = []
    for pic in X:
        blur = cv2.bilateralFilter(src=pic, d=9, sigmaColor=75, sigmaSpace=75)
        res.append(blur)
    return np.array(res)
noise_train = add_noise(X_train)
noise_test = add_noise(X_test)
```

Capítulo 4

Resultados

4.1. Desenfoque Gaussiano

Tomo una imagen para compararlos al aplicar

(a) original (b) desenfocado (c) predecido

Figura 4.1: Muestra aleatoria

Una muestra mas amplia de los resultados:

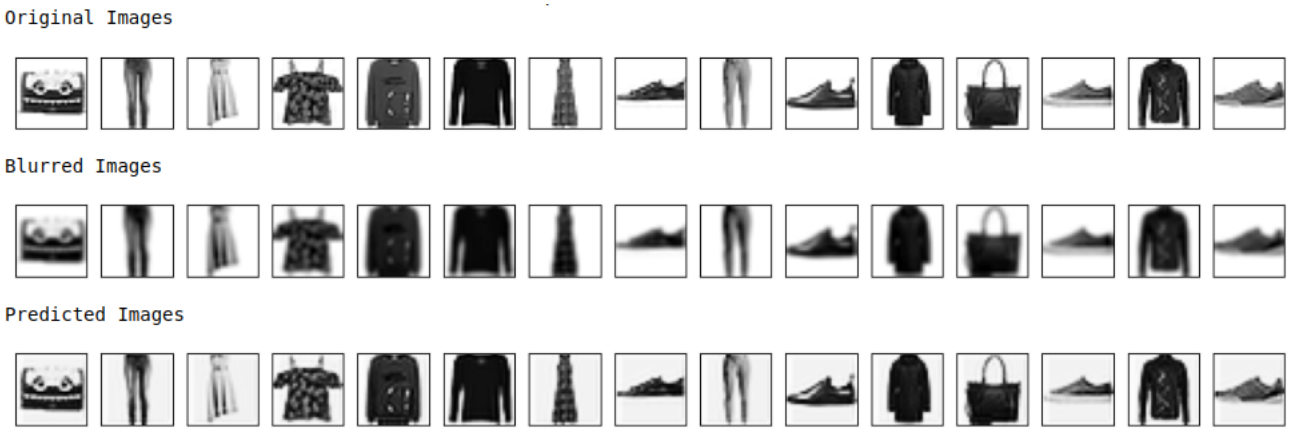


Figura 4.2: Muestras aleatorias

4.2. Desenfoque Promedio

Tomo una imagen para compararlos al aplicar

(a) original (b) desenfocado (c) predecido

Figura 4.3: Muestra aleatoria

Una muestra mas amplia de los resultados:

Figura 4.4: Muestras aleatorias

4.3. Desenfoque Medio

Tomo una imagen para compararlos al aplicar

(a) original (b) desenfocado (c) predecido

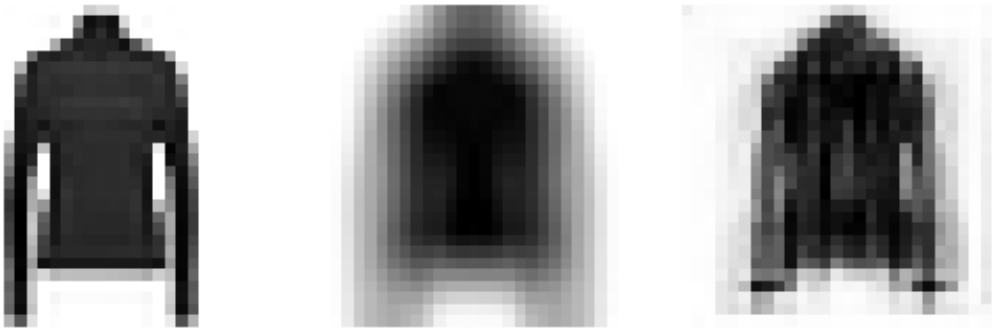
Figura 4.5: Múltiples imágenes

Una muestra mas amplia de los resultados:

Figura 4.6: Muestras aleatorias

4.4. Desenfoque de Filtrado Bilateral

Tomo una imagen para compararlos al aplicar



(a) original (b) desenfocado (c) predecido

Figura 4.7: Muestra aleatoria

Una muestra mas amplia de los resultados:

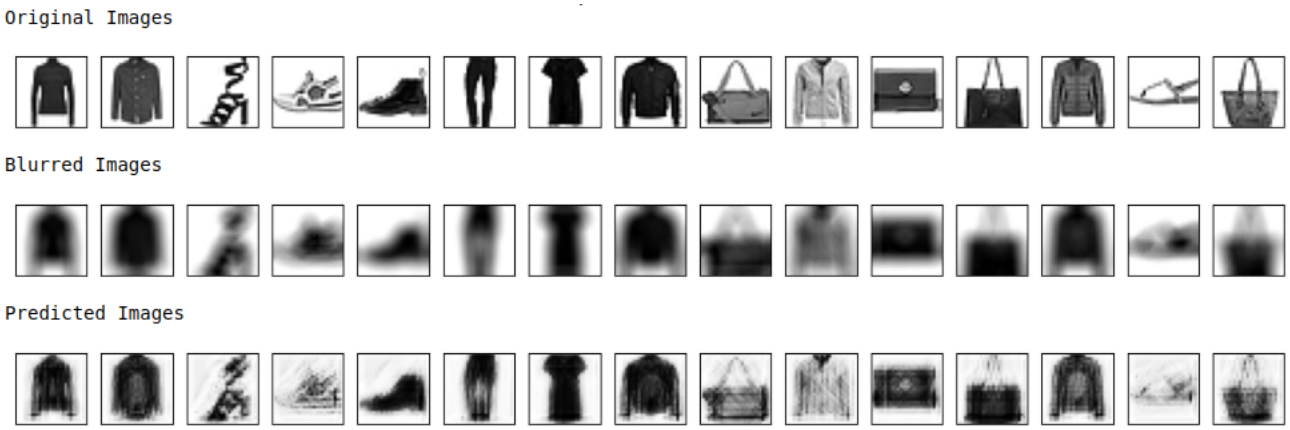


Figura 4.8: Muestras aleatorias

4.5. Eficiencia del Modelo

El porcentaje de error para cada técnica de desenfoque se muestra a continuación:

Modelo	Porcentaje Error
Gaussiano	1.65 %
Promedio	2.46 %
Medio	17.94 %
Filtrado Bilateral	8.46 %

Cuadro 4.1: Porcentaje de error

Capítulo 5

Conclusion y trabajos futuros

5.1. Conclusiones

Se logró desenfocar las imágenes mediante las técnicas propuestas; Se pudo observar que las imágenes predichas son similares a las imágenes originales debido a su bajo porcentaje de error, salvo el desenfocado Medio que salió alto. Por lo tanto el modelo entrenado fue correcto.

5.2. Trabajos futuros

El trabajo actual nos sirve para evaluar como este nuevo enfoques de aprendizaje profundo pueden resolver el problema de desenfoque de imágenes. Lo cual puede plantearse para implementación real. O estudiarse en un ambiente o contexto distinto. Y analizar las limitación para un contexto dinámico o de tiempo real.

Bibliografía

[1] Bootcamp AI. *Intro a las redes neuronales convolucionales*. URL: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8.

[2] *OpenCV - Smoothing Images*. URL: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[3] ZALANDO. *Fashion MNIST*. 2018. URL: https://www.kaggle.com/datasets/zalando-research/fashionmnist.

[4] Kaihao Zhang et al. *Deep Image Deblurring: A Survey*. 2022.

[5] Jannik Zürn. *But what is an Autoencoder?* 2019. URL: https://jannik-zuern.medium.com/but-what-is-an-autoencoder-26ec3386a2af.