PROYECTO FINAL

Red Neuronal

Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de ciencias computacionales

Prof: Profesor López Arce Delgado Jorge Ernesto

Alumnos: *Jesús Abraham Flores Guzmán*

Miguel Axel Guizar Gonzales

Carrera: Ingeniería Informática

Materia: i5885 Seminario de Algoritmia

NRC: 60091

Sección: D11

Calendario: 2023A

Contenido

¿Qué es una red neuronal?

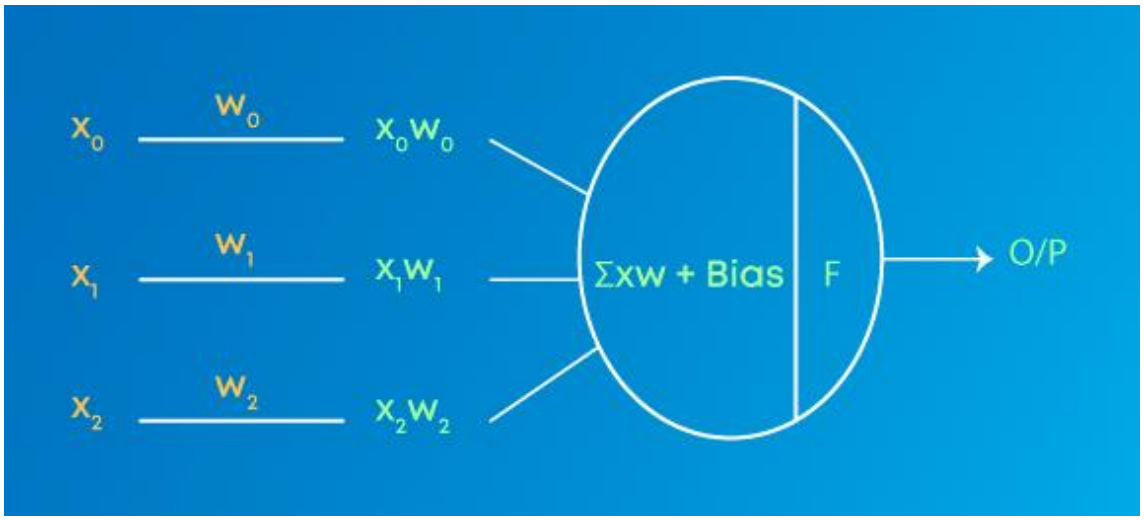
Una red neuronal es un método de la inteligencia artificial que enseña a las computadoras a procesar datos de una manera que está inspirada en la forma en que lo hace el cerebro humano. Se trata de un tipo de proceso de machine learning llamado aprendizaje profundo, que utiliza los nodos o las neuronas interconectados en una estructura de capas que se parece al cerebro humano. Crea un sistema adaptable que las computadoras utilizan para aprender de sus errores y mejorar continuamente. De esta forma, las redes neuronales artificiales intentan resolver problemas complicados, como la realización de resúmenes de documentos o el reconocimiento de rostros, con mayor precisión.

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.

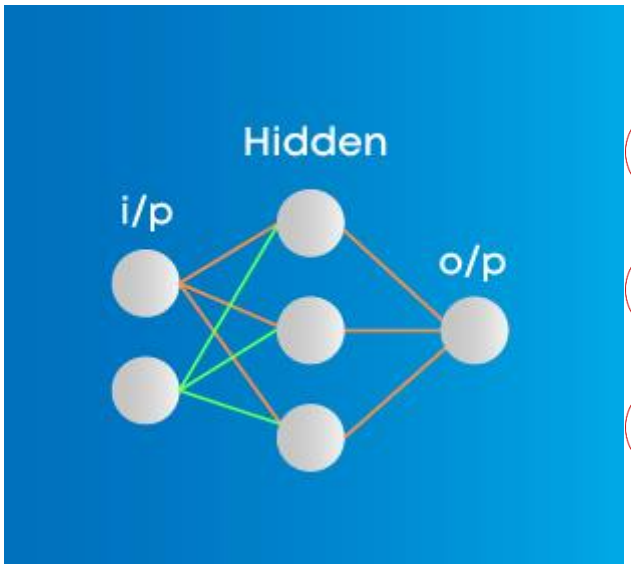
- Una neurona artificial está formada por:
- Un conjunto de entradas que son los enlaces o interconexiones por donde reciben información del exterior de la red o de otras neuronas. Cada una de las entradas está ponderada por un peso que determina la importancia de la información que recibe por esa interconexión y que es un valor que se va ajustando durante el entrenamiento de la red neuronal para minimizar el error de la red y que los resultados sean más fiables.
- Un conjunto de funciones:
    - Función de propagación: que relaciona matemáticamente los valores de las entradas, sus pesos y un sesgo (bias) para el cálculo del valor de salida de la neurona.
    - Función de activación: que determina si la neurona está activa o no, es decir, si produce un valor de salida o no.
    - Función de transferencia: que modula o adapta el valor calculado de salida de la neurona.
- La salida de la neurona que es el enlace o interconexión por donde entrega el resultado al exterior de la red neuronal u otras neuronas.

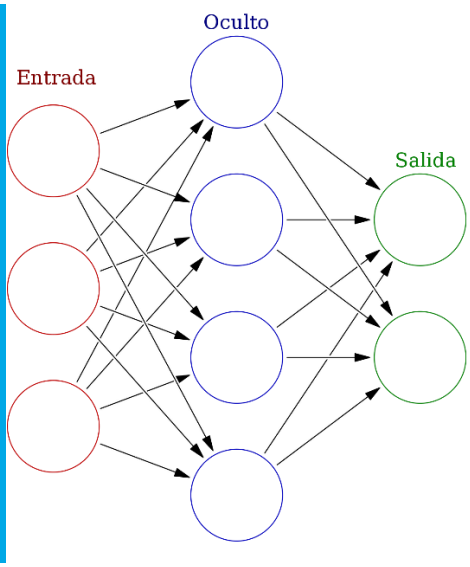
Cada neurona recibe una versión multiplicada de entradas y pesos aleatorios, que luego se agrega con un valor de sesgo estático (único para cada capa de neurona); esto luego se pasa a una función de activación apropiada que decide el valor final que se dará a la neurona.

Hay varias funciones de activación disponibles según la naturaleza de los valores de entrada. Una vez que se genera la salida a partir de la capa de red neuronal final, se calcula la función de pérdida (entrada frente a salida) y se realiza una retro propagación donde los pesos se ajustan para que la pérdida sea mínima. Encontrar los valores óptimos de los pesos es en lo que se centra la operación.



Los pesos (Weights) son valores numéricos que se multiplican por entradas. En retro propagación, se modifican para reducir la pérdida. En palabras simples, los pesos son valores aprendidos por máquina de las redes neuronales. Se autoajustan según la diferencia entre las salidas previstas y las entradas de entrenamiento.
La función de activación es una fórmula matemática que ayuda a la neurona a encenderse o apagarse.

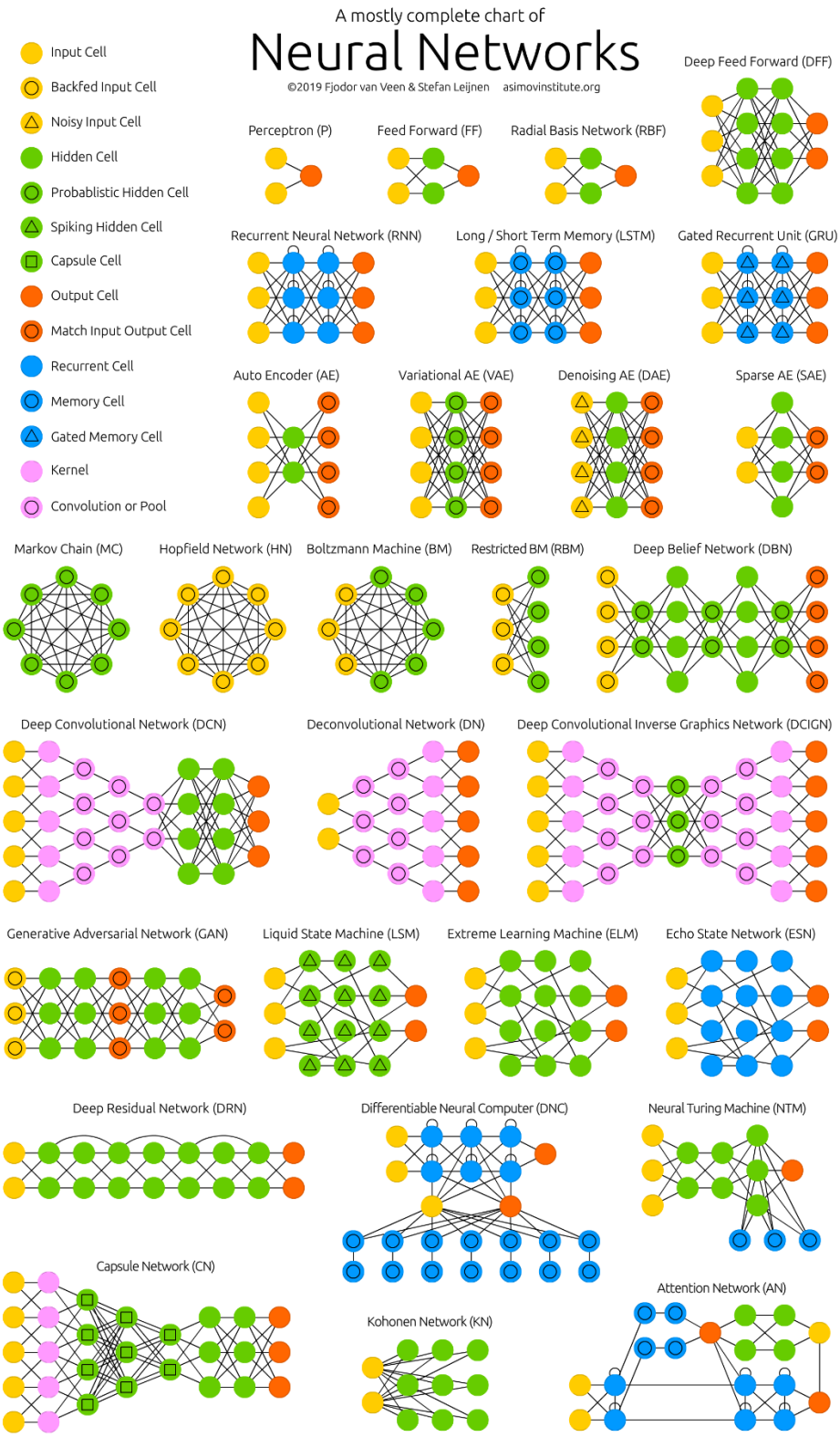




- **La capa de entrada** representa las dimensiones del vector de entrada.
- **La capa oculta** representa los nodos intermedios que dividen el espacio de entrada en regiones con límites (suaves). Toma un conjunto de entradas ponderadas y produce una salida a través de una función de activación.
- **La capa de salida** representa la salida de la red neuronal.

Tipos de redes neuronales

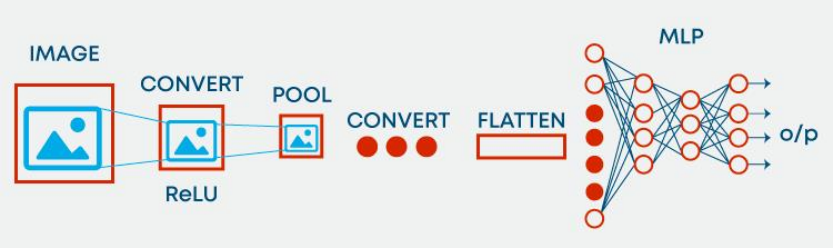
Hay muchos tipos de redes neuronales disponibles o que podrían estar en etapa de desarrollo. Se pueden clasificar según su: Estructura, Flujo de datos, Neuronas utilizadas y su densidad, Capas y sus filtros de activación de profundidad, etc.



Pero en este proyecto nos centramos en la red neuronal de convolución o red neuronal convolucional:

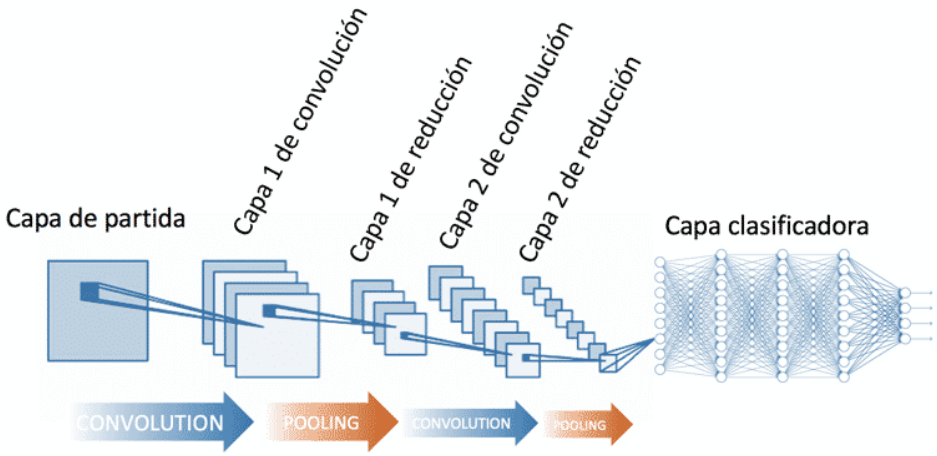
Red Neuronal Convolucional

La red neuronal de convolución contiene una disposición tridimensional de neuronas en lugar de la matriz bidimensional estándar. La primera capa se llama capa convolucional. Cada neurona en la capa convolucional solo procesa la información de una pequeña parte del campo visual. Las características de entrada se toman por lotes como un filtro. La red comprende las imágenes en partes y puede calcular estas operaciones varias veces para completar el procesamiento completo de la imagen. El procesamiento implica la conversión de la imagen de escala RGB o HSI a escala de grises. Avanzar en los cambios en el valor del píxel ayudará a detectar los bordes y las imágenes se pueden clasificar en diferentes categorías.



La propagación es unidireccional donde CNN contiene una o más capas convolucionales seguidas de agrupación y bidireccional donde la salida de la capa convolucional va a una red neuronal completamente conectada para clasificar las imágenes como se muestra en el diagrama anterior. Los filtros se utilizan para extraer ciertas partes de la imagen. En MLP, las entradas se multiplican con pesos y se alimentan a la función de activación. Convolution usa RELU y MLP usa la función de activación no lineal seguida de softmax. Las redes neuronales de convolución muestran resultados muy efectivos en el reconocimiento de imágenes y videos, el análisis semántico y la detección de paráfrasis.

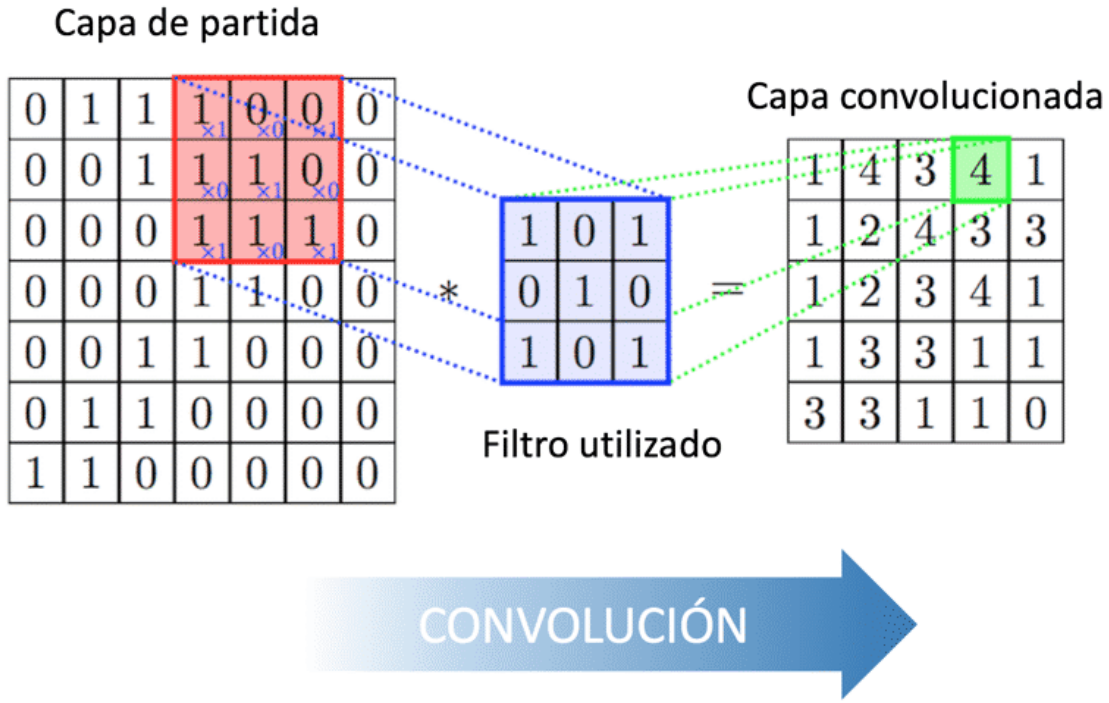
Esta red multicapa consta de capas convolucionales y de reducción alternadas, y finalmente tiene capas de conexión total como una red perceptrón multicapa.



Convolución

En la convolución se realizan operaciones de productos y sumas entre la capa de partida y los n filtros (o kernel) que genera un mapa de características. Las características extraídas corresponden a cada posible ubicación del filtro en la imagen original.

La ventaja es que el mismo filtro (neurona) sirve para extraer la misma característica en cualquier parte de la entrada, con esto que consigue reducir el número de conexiones y el número de parámetros a entrenar en comparación con una red multicapa de conexión total.



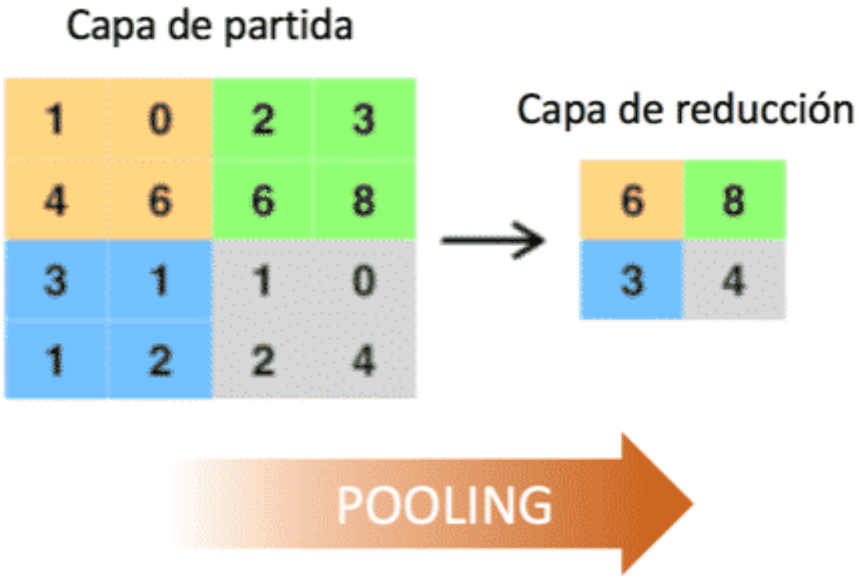
Después de aplicar la convolución a los mapas de características se les aplica una función de activación.

La función de activación recomendada es signoide ReLU, seleccionando una tasa de aprendizaje adecuada y monitorizando la fracción de neuronas muertas. También se podría probar con Leaky ReLu o Maxout, y la menos recomendada sería la sigmoide logística, la razón principal es que la sigmoide satura en los extremos, lo que puede llevar a gradientes cercanos a cero en esas regiones. Esto puede ralentizar el proceso de entrenamiento y dificultar la convergencia. Además, la sigmoide comprime los valores a un rango limitado, lo que limita la capacidad de representación de la red.

Pooling

En el pooling se disminuye la cantidad de parámetros al quedarse con las características más comunes.

La forma de reducir parámetros se realiza mediante la extracción de estadísticas como el promedio o el máximo de una región fija del mapa de características, al reducir características el método pierde precisión, aunque mejora su compatibilidad.



Clasificador

Al final de las capas convolucionales y de reducción, se suele utilizar capas completamente conectadas en la que cada píxel se considera como una neurona separada al igual que en un perceptrón multicapa.

La última capa de esta red es una capa clasificadora que

Ventajas de usar este tipo de red neuronal:

- Se utiliza para el aprendizaje profundo con pocos parámetros.
- Menos parámetros para aprender en comparación con la capa totalmente conectada

Desventajas:

- Comparativamente complejo de diseñar y mantener
- Comparativamente lento [depende del número de capas ocultas]

DATASET

Utilizar un conjunto de datos (dataset) es fundamental al entrenar y evaluar una red neuronal por varias razones importantes.

Aprendizaje

Las redes neuronales requieren una gran cantidad de datos para aprender patrones y características relevantes. Al proporcionar a la red un conjunto de datos diverso y representativo, se le permite aprender y generalizar a partir de esa información. Cuanto más grande y diverso sea el conjunto de datos, mayor será la capacidad de la red para capturar la complejidad de los datos y realizar predicciones precisas.

Generalización

El objetivo principal de entrenar una red neuronal es que pueda generalizar y realizar predicciones precisas sobre datos no vistos previamente. Al utilizar un conjunto de datos separado en conjuntos de entrenamiento y prueba, se puede evaluar cómo la red se desempeña en datos no utilizados durante el entrenamiento. Esto proporciona una medida objetiva de la capacidad de generalización de la red.

Evaluación del rendimiento

Un conjunto de datos adecuado permite evaluar y comparar el rendimiento de diferentes arquitecturas de red, hiperparámetros y técnicas de entrenamiento. Al utilizar métricas de evaluación como precisión, recall o F1-score, se puede cuantificar el desempeño de la red y realizar comparaciones objetivas entre diferentes enfoques.

Evitar el sobreajuste

El sobreajuste ocurre cuando una red neuronal se vuelve demasiado específica para el conjunto de datos de entrenamiento y no generaliza bien en nuevos datos. Al utilizar un conjunto de datos separado para la validación o prueba, se puede monitorear si la red está sobreajustando. Esto ayuda a tomar medidas correctivas, como el uso de técnicas de regularización, para mejorar el rendimiento y la capacidad de generalización.

Reproducibilidad y comparabilidad

El uso de un conjunto de datos estándar y ampliamente aceptado en un campo específico permite que otros investigadores reproduzcan los resultados y comparen sus enfoques directamente. Esto facilita la colaboración, la validación cruzada y el avance del conocimiento en el campo de las redes neuronales.

Para el proyecto utilizamos un dataset que encontramos en la pagina web KAGGLE el cual tiene 1070 archivos de imágenes de captchas en una escala de grises.

Las imágenes son palabras de 5 letras que pueden contener números. A las imágenes se les ha aplicado ruido (desenfoque y una línea) y tienen un tamaño de 200 x 50 PNG.

El conjunto de datos proviene de Wilhelmy, Rodrigo & Rosas, Horacio. (2013). conjunto de datos captcha.

Aquí algunos ejemplos:

Preprocesamiento de datos

Para el preprocesamiento de las imágenes usamos las siguientes librerías

```
import cv2

import numpy as np

import matplotlib.pyplot as plt
```

importamos las imágenes

```
path1 = './samples/23n88.png'

path2 = './samples/23mdg.png'

img1 = cv2.imread(path1, cv2.IMREAD_GRAYSCALE)

img2 = cv2.imread(path2, cv2.IMREAD_GRAYSCALE)
```

se leen dos imágenes utilizando la función cv2.imread() de la biblioteca OpenCV. Las imágenes se cargan desde los archivos especificados en path1 y path2.

cv2.imread() es una función de OpenCV que se utiliza para leer imágenes de archivos en formato de imagen. Toma dos argumentos principales:

- path1 y path2: Las rutas de los archivos de imagen que se desean leer.
- cv2.IMREAD_GRAYSCALE: Es una bandera que especifica cómo se debe cargar la imagen. En este caso, cv2.IMREAD_GRAYSCALE se utiliza para cargar la imagen en escala de grises, lo que significa que la imagen se leerá en blanco y negro, sin información de color.

Como resultado, img1 y img2 son matrices NumPy que contienen los píxeles de las imágenes en escala de grises.

```
thresh_img1 = cv2.adaptiveThreshold(img1, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 145, 0)

thresh_img2 = cv2.adaptiveThreshold(img2, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 145, 0)
```

Realiza una operación de umbral adaptativo en la imagen y almacena el resultado en la variable thresh_img.

- **img**: La imagen de entrada en la que se aplicará el umbral adaptativo.
- **255**: El valor máximo que se asignará a los píxeles que superen el umbral.
- **cv2.ADAPTIVE_THRESH_GAUSSIAN_C**: El método utilizado para calcular el umbral adaptativo. En este caso, se utiliza el método de

umbralización adaptativa gaussiana, que calcula el umbral para cada píxel como una ponderación de los píxeles vecinos utilizando una distribución gaussiana.

- **cv2.THRESH_BINARY**: El tipo de umbralización que se aplicará después de calcular el umbral adaptativo. En este caso, se utiliza la umbralización binaria, donde los píxeles que superen el umbral se establecen en el valor máximo (255) y los demás se establecen en cero.
- **145:** El tamaño de bloque utilizado para calcular el umbral adaptativo. Especifica el tamaño de la vecindad que se utilizará para calcular el umbral de cada píxel. Un valor mayor implicará una adaptación a una escala de vecindad más grande.
- **0:** Un valor constante que se resta del resultado final del umbral adaptativo. En este caso, no se resta ningún valor constante.

close_img1 = cv2.morphologyEx(thresh_img1, cv2.MORPH_CLOSE, np.ones((5,2), np.uint8))

close_img2 = cv2.morphologyEx(thresh_img2, cv2.MORPH_CLOSE, np.ones((5,2), np.uint8))

Se aplica una operación de cierre morfológico a las imágenes que es una operación de procesamiento de imágenes que combina dos operaciones básicas: dilatación y erosión.

1. thresh_img1 y thresh_img2: Las imágenes binarias en las que se aplicará la operación morfológica de cierre.
2. cv2.MORPH_CLOSE: Un indicador que especifica el tipo de operación morfológica que se realizará, en este caso, el cierre morfológico.
3. np.ones((5,2): Un kernel o matriz de estructuración que define la forma y el tamaño del elemento estructurante utilizado en la operación. En este caso, se utiliza una matriz de unos de tamaño 5x2.
4. Np.uint8: Es un tipo de dato numérico sin signo de 8 bits en NumPy. Se refiere a una matriz de enteros sin signo de 8 bits, lo que significa que los valores que puede contener van desde 0 hasta 255. Se utiliza comúnmente para representar imágenes en escala de grises

dilate_img1 = cv2.dilate(close_img1, np.ones((2,2), np.uint8), iterations = 1)

dilate_img2 = cv2.dilate(close_img2, np.ones((2,2), np.uint8), iterations = 1)

La dilatación es una operación morfológica utilizada para expandir las regiones o contornos de los objetos presentes en una imagen.

En OpenCV, la función cv2.dilate() se utiliza para aplicar la operación de dilatación a una imagen. Toma varios argumentos:

- **close_img1 y close_img2:** Las imágenes en las que se aplicará la operación de dilatación.
- **np.ones((2,2), np.uint8):** El elemento estructurante utilizado en la operación de dilatación. En este caso, se utiliza una matriz de unos de tamaño 2x2 como elemento estructurante.
- **iterations=1:** El número de veces que se aplicará la dilatación. En este caso, se realiza solo una iteración.

El elemento estructurante, definido como np.ones((2,2), np.uint8), es una matriz cuadrada compuesta por unos que especifica la forma y el tamaño de la dilatación. En este caso, es una matriz de 2x2, lo que significa que los objetos en la imagen se expandirán en todas las direcciones por un píxel.

gauss_img1 = cv2.GaussianBlur(dilate_img1, (1,1), 0)

gauss_img2 = cv2.GaussianBlur(dilate_img2, (1,1), 0)

El suavizado gaussiano es una técnica de filtrado utilizada para reducir el ruido en una imagen y suavizar las transiciones abruptas entre los píxeles.

En OpenCV, la función cv2.GaussianBlur() se utiliza para aplicar el suavizado gaussiano a una imagen. Toma varios argumentos:

- dilate_img1 y dilate_img2: Las imágenes en las que se aplicará el suavizado gaussiano.
- (1,1): El tamaño del kernel o ventana de suavizado gaussiano. En este caso, se utiliza un kernel de tamaño 1x1, lo que significa que se tomará en cuenta solo el píxel central para calcular el valor suavizado.
- 0: La desviación estándar en la dirección x e y. En este caso, se establece en 0, lo que indica que la desviación estándar se calculará automáticamente en función del tamaño del kernel.

El suavizado gaussiano se basa en una función de distribución gaussiana para calcular el valor suavizado de cada píxel en función de su vecindario. La función de distribución gaussiana pondera los píxeles vecinos más cercanos al píxel actual, asignándoles un mayor peso en el cálculo del valor suavizado.

cv2.rectangle(gauss_img1, (30,12), (50,49), 0, 1)

En esta parte del código se están dibujando rectángulos en las imágenes utilizando la función cv2.rectangle(). Cada línea de código corresponde a un rectángulo diferente que se dibuja en la imagen.

Donde:

- image: La imagen en la cual se va a dibujar el rectángulo.
- pt1: Las coordenadas del punto de inicio del rectángulo (esquina superior izquierda).
- pt2: Las coordenadas del punto final del rectángulo (esquina inferior derecha).
- color: El color del rectángulo. En este caso, se utiliza el valor 0, que representa el color negro.
- thickness: El grosor del contorno del rectángulo. Se establece en 1 en este caso.

Arquitectura: Secuencial

La arquitectura secuencial en redes neuronales se refiere a un tipo de estructura de red neuronal en la cual las capas se apilan secuencialmente una tras otra, formando una cadena lineal. Es una forma sencilla y común de diseñar y construir redes neuronales.

En una arquitectura secuencial, cada capa de la red se conecta directamente a la capa anterior y a la capa siguiente, sin saltos ni conexiones adicionales. La información fluye de manera secuencial a través de las capas, de ahí el nombre de "arquitectura secuencial".

En una red neuronal secuencial típica, las capas pueden tener diferentes funciones y características, como capas convolucionales, capas de agrupación (pooling), capas de activación, capas completamente conectadas, entre otras. Estas capas se organizan en un orden específico para construir la arquitectura secuencial.

La arquitectura secuencial es ampliamente utilizada y es especialmente común en modelos como las redes neuronales convolucionales (CNN) para la visión por computadora, donde se utilizan capas convolucionales y capas de agrupación para extraer características de las imágenes. También se utiliza en modelos como las redes neuronales recurrentes (RNN) para el procesamiento de secuencias, donde las capas recurrentes se apilan secuencialmente para procesar la información en secuencia.

Capaz convolucional

Para el proyecto usamos 2 capas de keras con varios filtros cada una:

Conv2d

Esta capa crea un núcleo de convolución que se convoluciona con la entrada de la capa para producir un tensor de salidas. Si use_bias es True, se crea un vector de sesgo y se agrega a las salidas. Finalmente, si la activación no es Ninguna, también se aplica a las salidas.

Dense

Dense implementa la operación: salida = activación (punto (entrada, núcleo) + sesgo) donde la activación es la función de activación por elementos que se pasa como argumento de activación, el núcleo es una matriz de pesos creada por la capa y el sesgo es un vector de sesgo creado por la capa (solo aplicable si use_bias es True).

Cada una cuenta con cierto numero de filtros que se le agregan con esa función

```
model = cnn(128, 32, 16, 32, 32)
model.summary()
```

Y estos serian los filtros

```python
def conv_layer(filterx):
    model = Sequential()

    model.add(Conv2D(filterx, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))  # previene el sobre apendizaje
    model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

    return model


def dens_layer(hiddenx):
    model = Sequential()

    model.add(Dense(hiddenx, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))

    return model


def cnn(filter1, filter2, filter3, hidden1, hidden2):
    model = Sequential()
    model.add(Input((40, 20, 1,)))

    model.add(conv_layer(filter1))
    model.add(conv_layer(filter2))
    model.add(conv_layer(filter3))

    model.add(Flatten())
    model.add(dens_layer(hidden1))
    model.add(dens_layer(hidden2))

    model.add(Dense(19, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model
```

Funciones de activación

A la salida de la neurona, puede existir, un filtro, función limitadora o umbral, que modifica el valor resultado o impone un límite que se debe sobrepasar para poder proseguir a otra neurona. Esta función se conoce como función de activación.

Una función de activación es, por tanto, una función que transmite la información generada por la combinación lineal de los pesos y las entradas, es decir son la manera de transmitir la información por las conexiones de salida. La información puede transmitirse sin modificaciones, función identidad, o bien que no transmita la información. Como el objetivo es que la red neuronal sea capaz de resolver problemas cada vez más complejos, las funciones de activación generalmente harán que los modelos sean no lineales.

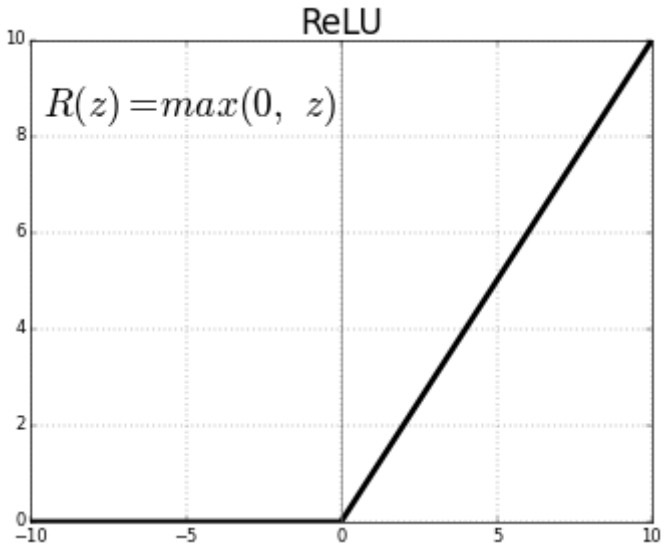
Función Rectificadora (ReLU).

ReLU es un acrónimo de Rectified Linear Unit. La ReLU se define como:

$$ReLU(x) = max(0, x)$$

$$ReLU'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & 0 < x \end{cases}$$

Básicamente, dado un valor x si dicho valor es menor que 0, conviértelo a 0, y si es mayor, devuelve el mismo valor. Esto en el Deep Learning se traduce como “siempre que una neurona de un Output negativo, desactiva dicha neurona”. Esta es su gráfica:



Propiedades matemáticas:

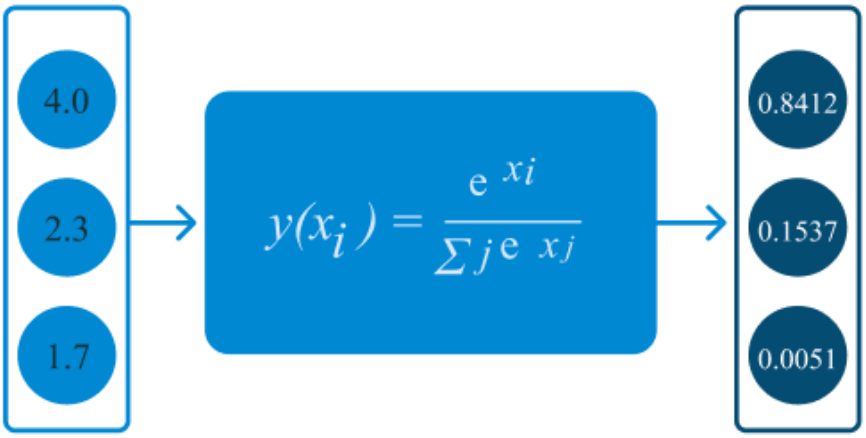
- Es continua
- NO lineal
- NO derivable
- Va de 0 a ∞

Una de las grandes ventajas de esta función es su simplicidad y por tanto rapidez computacional, lo que en los procesos de Deep Learning que pueden durar horas/días/semanas, es una gran ventaja. Su código sería así:

```
def relu(x):
    return max(0, i)
```

Softmax

Softmax no es una función de activación propiamente dicha, en el sentido que no recibe como entrada un único elemento, sino un vector. Realiza la exponencial de cada elemento y la divide entre la suma de las exponenciales del vector. El resultado es que esta función de activación traduce las salidas de la capa previa a un vector que simula la distribución de probabilidades de pertenencia a las diferentes clases:



Para esta versión de la Softmax, las categorías son excluyentes, es decir, si se asigna una categoría, ya no se puede pertenecer a otra.

Normalmente esta función es usada en la capa final de la red neuronal, para realizar la predicción de dicha red neuronal.

En resumen, utilizamos la función softmax para que los valores obtenidos a partir de una transformación lineal sean interpretados como probabilidades de que determinada entrada pertenezca a una clase en particular

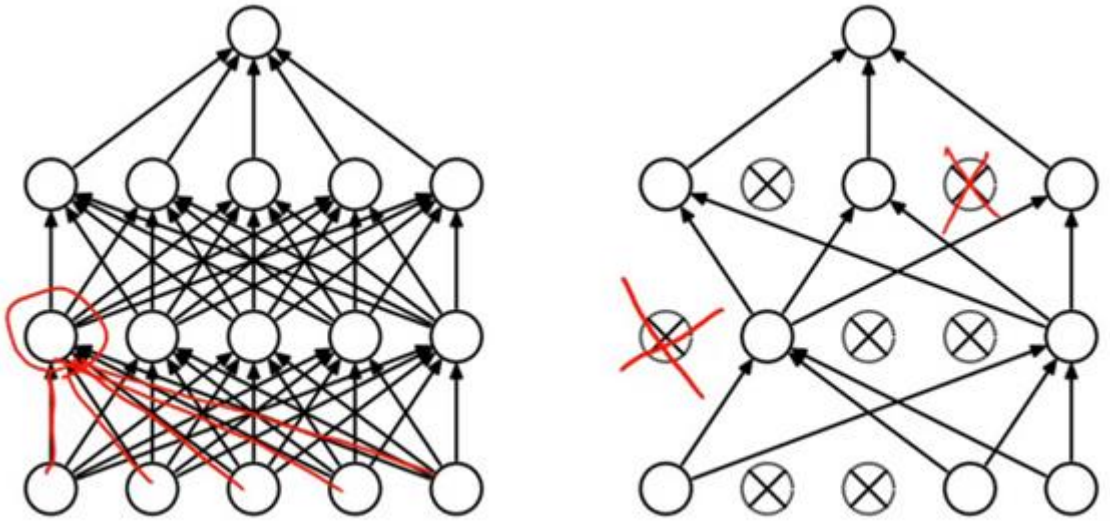
Técnicas de regularización

La regularización es una técnica que reduce el error de un modelo evitando el sobreajuste y entrenando al modelo para que funcione correctamente. consiste en penalizar de alguna forma las predicciones que hace nuestra red durante el entrenamiento, de forma que no piense que el *training set* es la verdad absoluta y así sepa generalizar mejor cuando ve otros datasets.

En este proyecto utilizamos:

Dropout

El procedimiento es sencillo: por cada nueva entrada a la red en fase de entrenamiento, se desactivará aleatoriamente un porcentaje de las neuronas en cada capa oculta, acorde a una **probabilidad de descarte** previamente definida. Dicha probabilidad puede ser igual para toda la red, o distinta en cada capa.



(a) Standard Neural Net (b) After applying dropout.

Lo que se consigue con esto es que ninguna neurona memorice parte de la entrada; que es precisamente lo que sucede cuando tenemos sobreajuste.

Una vez tengamos el modelo listo para realizar predicciones sobre muestras nuevas, debemos compensar de alguna manera el hecho de que no todas las neuronas permanecieran activas en entrenamiento, ya que en inferencia sí que estarán todas funcionando y por tanto habrá más activaciones contribuyendo a la salida de la red. Un ejemplo de dicha compensación podría ser multiplicar todos los parámetros por la probabilidad de no descarte.

Aumento de los datos (Data augmentation)

El aumento de datos es una técnica para aumentar artificialmente el conjunto de entrenamiento mediante la creación de copias modificadas de un conjunto de datos utilizando datos existentes. Incluye realizar cambios menores en el conjunto de datos o usar el aprendizaje profundo para generar nuevos puntos de datos.

Ejemplos de transformaciones son:

- Voltear la imagen en horizontal / vertical
- Rotar la imagen X grados.
- Recortar, añadir relleno, redimensionar, …
- Aplicar deformaciones de perspectiva
- Ajustar brillo, contraste, saturación, …
- Introducir ruido, defectos, …

Batch

Batch Normalization, también conocido como "normalización por lotes", es una técnica utilizada en el campo del aprendizaje automático y la inteligencia artificial para mejorar el rendimiento y la estabilidad de las redes neuronales artificiales durante el entrenamiento.

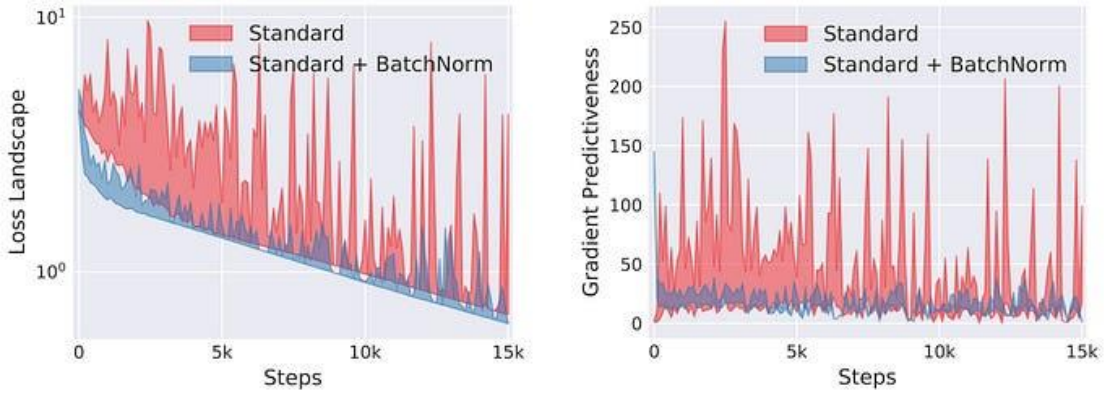
Durante el proceso de entrenamiento de una red neuronal, es común que los datos de entrada se normalicen para facilitar el procesamiento y mejorar la convergencia. Sin embargo, a medida que los datos se propagan a través de múltiples capas de una red neuronal profunda, los valores de activación pueden fluctuar y cambiar significativamente, lo que dificulta el entrenamiento efectivo.

La normalización por lotes aborda este problema normalizando las activaciones de cada capa para que tengan una media cercana a cero y una varianza cercana a uno. Esto se logra calculando la media y la varianza de las activaciones dentro

de un lote de datos y utilizando estas estadísticas para normalizar las activaciones.

Consiste básicamente en añadir un paso extra, habitualmente entre las neuronas y la función de activación. Lo ideal es que la normalización se hiciera usando la media y la varianza de todo el conjunto de entrenamiento, pero si estamos aplicando el descenso del gradiente estocástico para entrenar la red, se usará la media y la varianza de cada mini-lote de entrada.

Después se añaden 2 parámetros: un bias como sumando, y otra constante similar a un bias pero que aparece multiplicando cada activación. Esto se hace para que el rango de la entrada escale fácilmente hasta el rango de salida, lo que ayudará mucho a nuestra red a la hora de ajustar a los datos de entrada, y reducirá las oscilaciones de la función de coste. Como consecuencia de esto podremos aumentar la tasa de aprendizaje y la convergencia hacia el mínimo global se producirá más rápidamente.



Imagina que estás enseñando a una red neuronal a reconocer imágenes de gatos y perros. Durante el entrenamiento, la red ve muchas imágenes una tras otra en pequeños grupos llamados lotes, a medida que las imágenes pasan por muchas capas en la red, los valores de activación pueden cambiar mucho y hacer que el entrenamiento sea más difícil, lo que hace esto es ajustar los valores de las activaciones en cada capa de la red para que estén en un rango más adecuado. Al normalizar las activaciones, la red neuronal puede aprender más rápido y de manera más estable.

Funcion de perdida Cross Entropy

La función de pérdida Cross Entropy (entropía cruzada) es una medida comúnmente utilizada en el aprendizaje automático para evaluar la discrepancia entre la distribución de probabilidad predicha por un modelo y la distribución de probabilidad real de los datos de entrenamiento.

Se utiliza principalmente en problemas de clasificación, donde se busca asignar una clase a un conjunto de datos de entrada. Esta función es especialmente útil cuando se trata de problemas de clasificación con múltiples clases

Es una función que penaliza solo las predicciones correctas, por lo que la función de perdidas solo es calculada para las predicciones correctas. Esta es la fórmula.

$$H(p, q) = -[p * \log(q) + (1 - p) * \log(1 - q)]$$

Donde:

- p representa la probabilidad real de la clase verdadera.
- q representa la probabilidad predicha por el modelo para la clase verdadera.

Para el cálculo de esta perdida se utiliza un vector **one-hot**, que en pocas palabras, es un vector de todos ceros, excepto la clase predicha que es uno. Por ejemplo, esta sería una predicción.

```
                        clase1 clase2 clase3 clase4
Etiqueta verdadera:        1      0      0      0
Predicciones softmax:     0.1    0.4    0.2    0.3
```

Como hemos dicho esta fórmula solo se aplica para la clase correcta por lo que tendríamos lo siguiente, con este ejemplo:

Cross Entropy Loss = -(1*log(0.1) + 0 + 0 + 0 = -log(0.1) = 2.303

Nuestro error es sería muy grande, porque le hemos dado muy baja probabilidad a la clase 1. Podemos hacer otro ejemplo:

```
                        clase1 clase2 clase3 clase4
Etiqueta verdadera:        1      0      0      0
Predicciones softmax:     0.9    0.02   0.04   0.04

Cross Entropy Loss = -(1 log(0.9) + 0 + 0 + 0) = -log(0.9) = 0.04
```

En este caso como predecimos la etiqueta 1 con un 90% de probabilidad, nuestro error es bajo.

One Hot Encoding

Esta estrategia consiste en crear una columna binaria (que solo puede contener los valores 0 o 1) para cada valor único que exista en la variable categórica que

estamos codificando, y marcar con un 1 la columna correspondiente al valor presente en cada registro, dejando las demás columnas con un valor de 0.

Por ejemplo, en el caso de la variable "sex" de los pasajeros del Titanic, One Hot Encoding crearía dos columnas binarias (una para el valor "male" y otra para el valor "female"). Para cada pasajero, se asignaría un valor de 1 a la columna correspondiente a su género y un valor de 0 a la columna del género opuesto. De esta manera, cada registro queda representado por un vector binario que indica la presencia o ausencia de cada valor categórico.

Una desventaja de este método es que estamos aumentando la dimensionalidad del conjunto de datos (es decir, aumentando el número de columnas o características categóricas a partir de las cuales entrenar el modelo), lo que puede resultar problemático si el número de muestras de las que se dispone no es suficientemente elevado.

Resultados

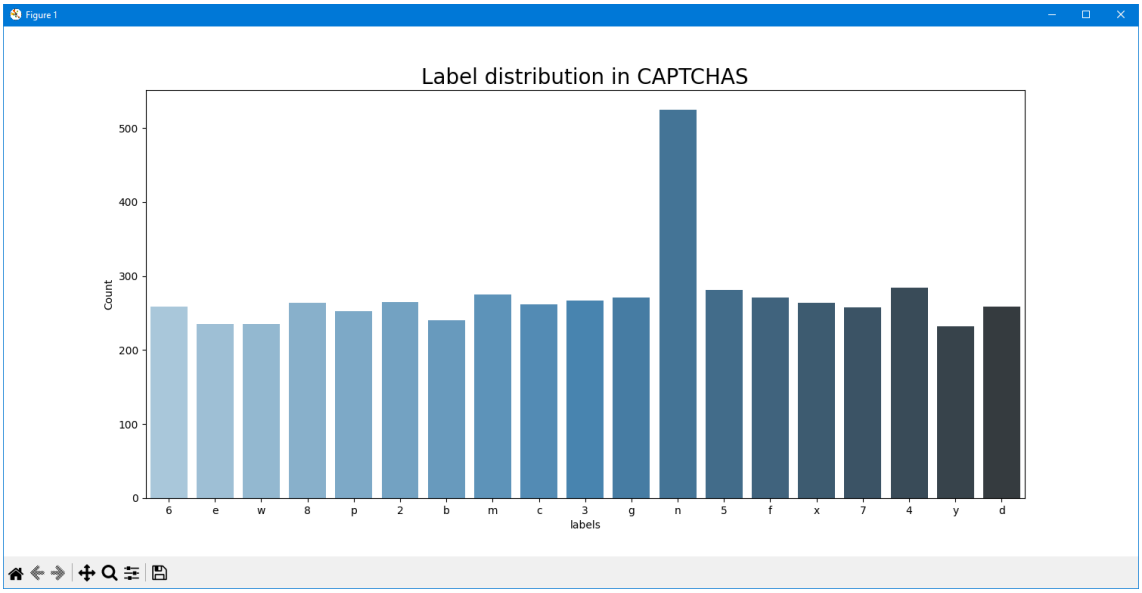


Figure 1
Label distribution in CAPTCHAS
Count
labels
0
100
200
300
400
500
6 e w 8 p 2 b m c 3 g n 5 f x 7 4 y d

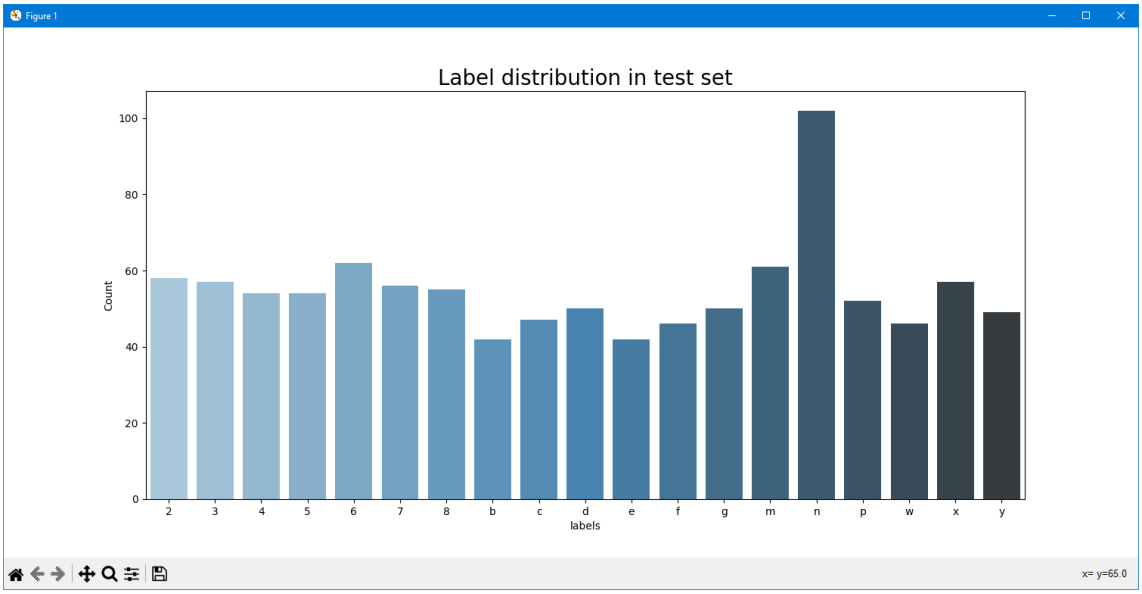
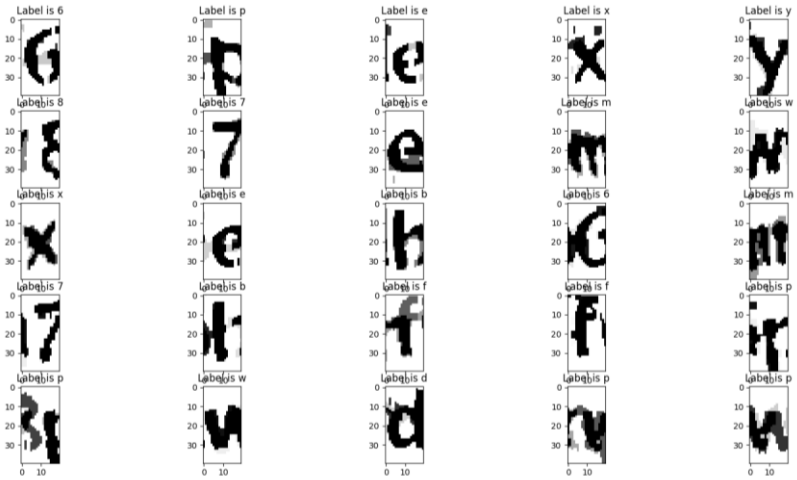
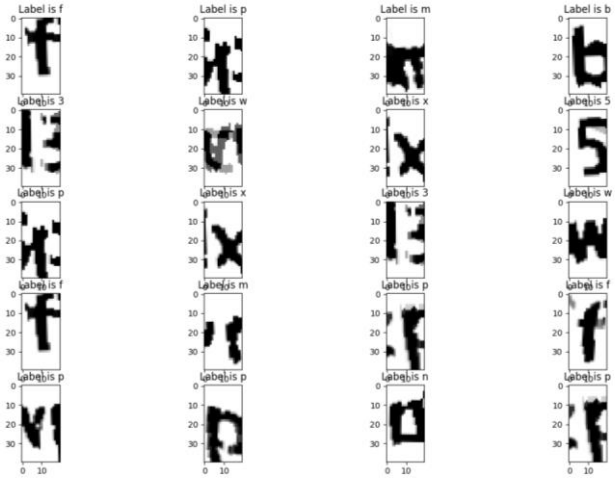
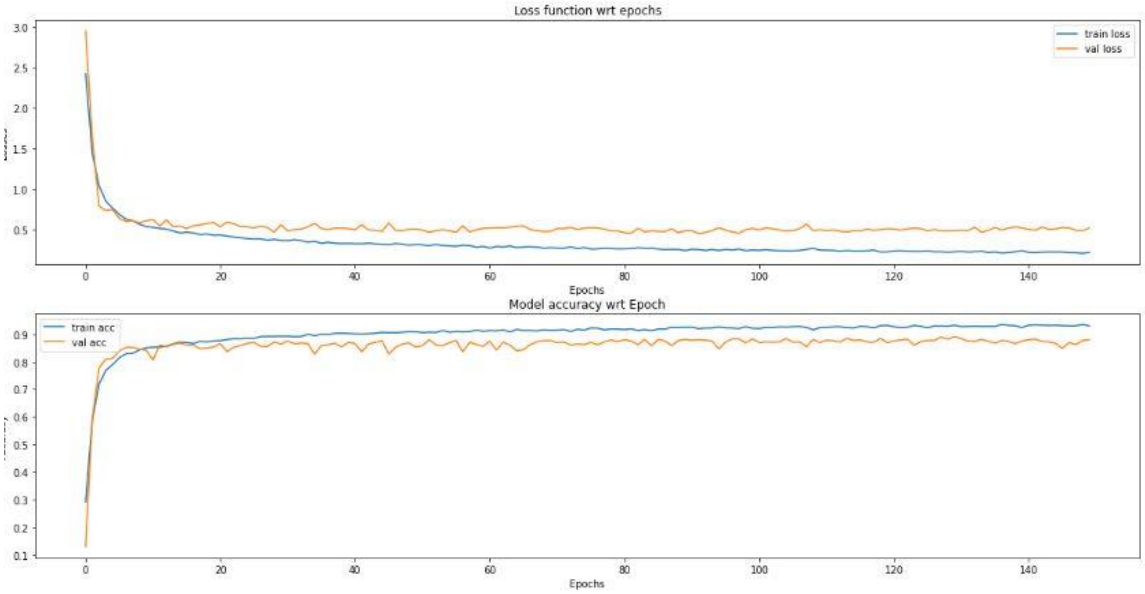


Figure 1
Label distribution in test set
Count
labels
0
20
40
60
80
100
2 3 4 5 6 7 8 b c d e f g m n p w x y
x= y=65.0


Label is 6
Label is p
Label is e
Label is x
Label is y
Label is 8
Label is 7
Label is e
Label is m
Label is w
Label is x
Label is e
Label is b
Label is 6
Label is m
Label is 7
Label is b
Label is f
Label is f
Label is p
Label is p
Label is w
Label is d
Label is p
Label is p





Accuracy : 0.8798076923076923

	precision	recall	f1-score	support
2	0.88	0.95	0.91	61
3	0.94	0.94	0.94	48
4	0.94	0.94	0.94	48
5	1.00	0.96	0.98	48
6	0.92	0.93	0.92	59
7	0.96	0.83	0.89	58
8	0.96	0.92	0.94	51
b	0.94	0.98	0.96	49
c	0.89	0.79	0.84	52
d	0.75	0.92	0.83	52
e	0.90	0.80	0.85	35
f	0.90	0.95	0.93	59
g	0.97	0.93	0.95	60
m	0.69	0.53	0.60	66
n	0.76	0.85	0.80	111
p	0.87	0.84	0.85	49
w	0.83	0.91	0.87	47
x	0.91	0.91	0.91	45
y	0.95	0.95	0.95	42
accuracy			0.88	1040
macro avg	0.89	0.89	0.89	1040
weighted avg	0.88	0.88	0.88	1040

Referencias

Alberto, R. (16 de 10 de 2020). *Medium*. Obtenido de https://rubialesalberto.medium.com/explicaci%C3%B3n-funciones-de-activaci%C3%B3n-y-pr%C3%A1ctica-con-python-5807085c6ed3

Amazon. (s.f.). *aws amazon*. Obtenido de https://aws.amazon.com/es/what-is/neural-network/

Calvo, D. (20 de 07 de 2017). *DiegoCalvo*. Obtenido de https://www.diegocalvo.es/red-neuronal-convolucional/

MyGreatLearning. (23 de 11 de 2022). Obtenido de https://www.mygreatlearning.com/blog/types-of-neural-networks/

Revista, U. (03 de 08 de 2021). *UNIR*. Obtenido de https://www.unir.net/ingenieria/revista/redes-neuronales-artificiales/

Saha, S. (15 de 12 de 2018). *SaturnCloud*. Obtenido de https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

Sitio Bid Data. (22 de 06 de 2019). Obtenido de https://sitiobigdata.com/2019/06/22/relu-funciones-activacion/

Soberanis, M. C. (17 de 12 de 2020). *Medium*. Obtenido de https://medium.com/soldai/introducci%C3%B3n-al-deep-learning-i-funciones-de-activaci%C3%B3n-b3eed1411b20