INTRODUCCIÓN A EMOTION AI

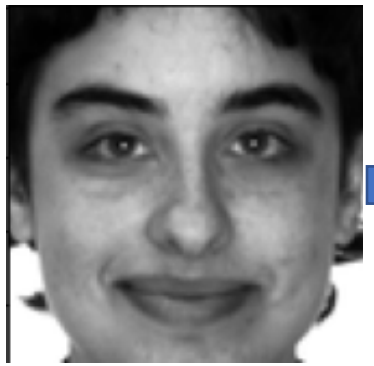
- La inteligencia artificial emocional (o Emotion AI) es una rama de la IA que permite que los ordenadores entiendan el lenguaje no verbal humano como las posturas corporales o expresiones faciales.
- Affectiva ofrece tecnología puntera en IA emocional: https://www.affectiva.com/

Crédito de la foto: https://en.wikipedia.org/wiki/File:11vRwNSw.jpg
Crédito de la foto: https://www.flickr.com/photos/jurvetson/49352718206

DESCRIPCIÓN DEL PROYECTO

- El objetivo de este proyecto es clasificar las emociones de las personas en función de sus imágenes faciales.
- En este caso práctico, asumiremos que trabajamos como consultores de IA / ML.
- Una empresa emergente de San Diego nos ha contratado para construir, entrenar e implementar un sistema que monitoriza automáticamente las emociones y expresiones de las personas.
- El equipo ha recopilado más de 20000 imágenes faciales, con sus etiquetas de expresión facial asociadas y alrededor de 2000 imágenes con sus anotaciones faciales de puntos clave.

IMAGEN ORIGINAL DE ENTRADA



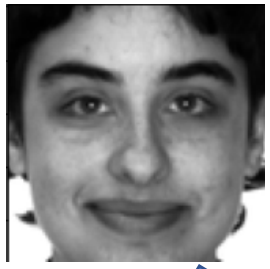
MODELO EMOTION AI

PREDICCIÓN DEL MODELO #1: EXPRESIÓN FACIAL (EMOCIÓN) FELICIDAD

PREDICCIÓN DEL MODELO #2: PUNTOS FACIALES CLAVE

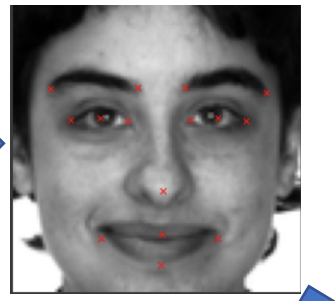
DESCRIPCIÓN DEL PROYECTO: FLUJO PARA DETECTAR PUNTOS CLAVE Y EMOCIONES

IMAGEN ENTRADA



1

MODELO DE DETECCIÓN PUNTOS FACIALES CLAVE



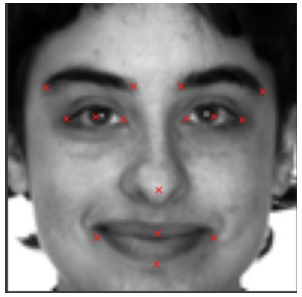
2

MODELO DE DETECCIÓN DE EXPRESIÓN FACIAL (EMOCIÓN)

CATEGORÍA EMOCIÓN: FELICIDAD

MODELO COMBINADO

EMOCIÓN = FELICIDAD



PARTE 1. DETECCIÓN DE PUNTOS FACIALES CLAVE

- **En la parte 1, crearemos un modelo de aprendizaje profundo basado en la red neuronal convolucional y los bloques residuales para predecir los puntos clave faciales.**

Data Source: https://www.kaggle.com/c/facial-keypoints-detection/data

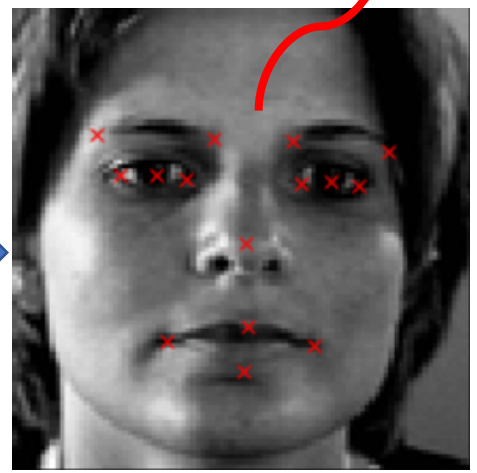
PARTE 1. DETECCIÓN DE PUNTOS FACIALES CLAVE

- El conjunto de datos consta de coordenadas x e y de 15 puntos clave faciales.
- Las imágenes de entrada son de 96 x 96 píxeles.
- Las imágenes constan de un solo canal de color (imágenes en escala de grises).

IMAGEN DE ENTRADA

MODELO DE DETECCIÓN DE PUNTOS FACIALES CLAVE

COORDENADAS DE LOS PUNTOS FACIALES CLAVE

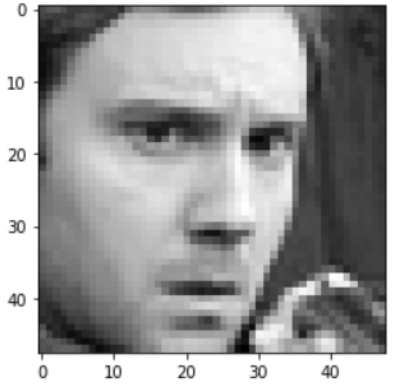


X_1
Y_1
X_2
Y_2
X_3
Y_3
.
.

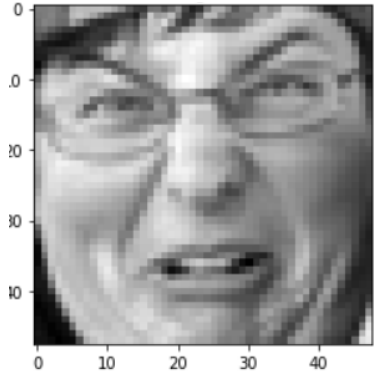
PARTE 2. DETECCIÓN DE EXPRESIÓN FACIAL (EMOCIÓN)

- El segundo modelo clasificará la emoción de las personas.
- Los datos contienen imágenes que pertenecen a 5 categorías:
  - 0 = ira
  - 1 = odio
  - 2 = tristeza
  - 3 = felicidad
  - 4 = sorpresa

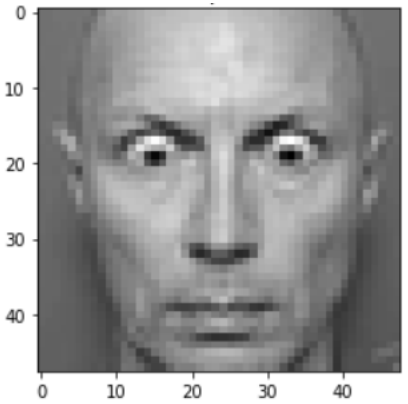
IRA



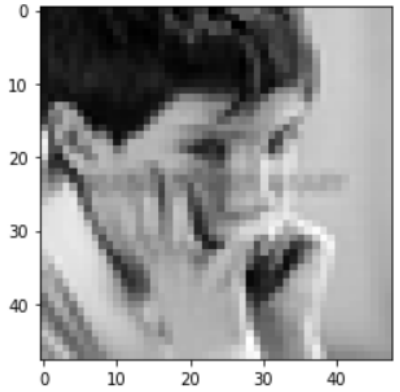
ODIO



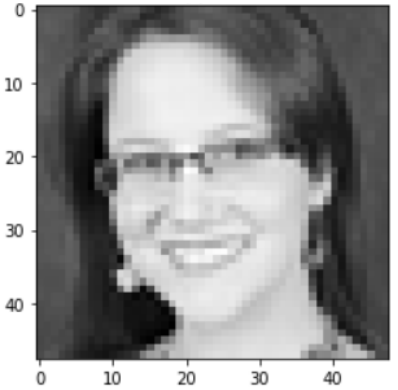
SORPRESA!



TRISTEZA



FELICIDAD



Los datos son originales de Kaggle: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data

PARTE 2. DETECCIÓN DE EXPRESIÓN FACIAL (EMOCIÓN)

IMAGEN DE ENTRADA

CLASIFICADOR (MODELO DE DEEP LEARNING)

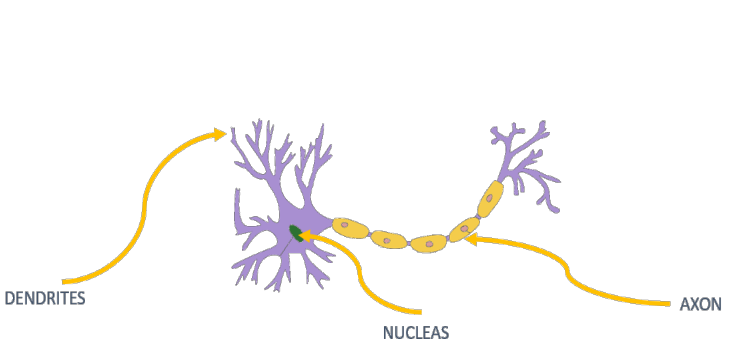
CLASES OBJETIVO

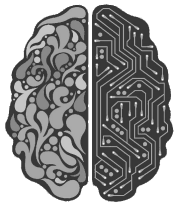
Ira
Odio
Tristeza
Felicidad
Sorpresa

MODELO MATEMÁTICO DE LA NEURONA

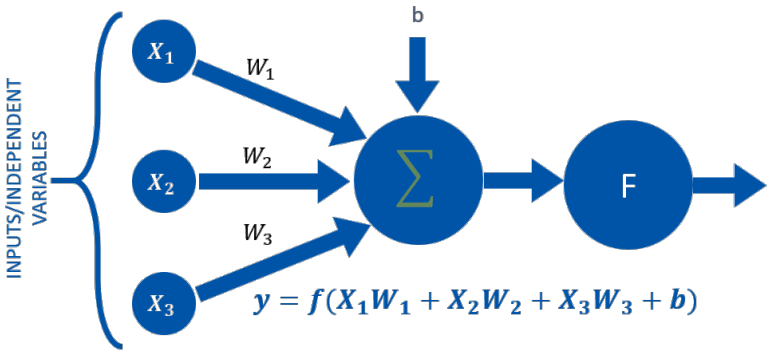
- El cerebro tiene más de 100 mil millones de neuronas que se comunican a través de señales eléctricas y químicas. Las neuronas se comunican entre sí y nos ayudan a ver, pensar y generar ideas.
- El cerebro humano aprende creando conexiones entre estas neuronas. Las RNA son modelos de procesamiento de información inspirados en el cerebro humano.
- La neurona recolecta señales de los canales de entrada llamados dendritas, procesa la información en su núcleo y luego genera una salida en una rama larga y delgada llamada axón.

NEURONA HUMANA





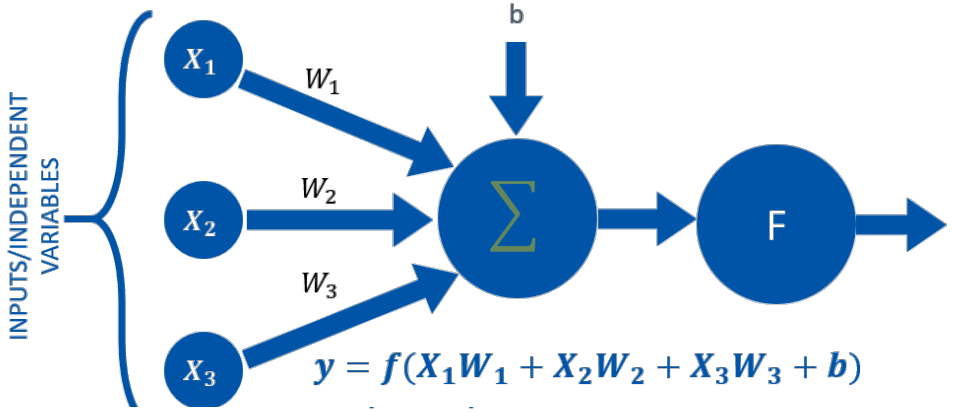
NEURONA ARTIFICIAL

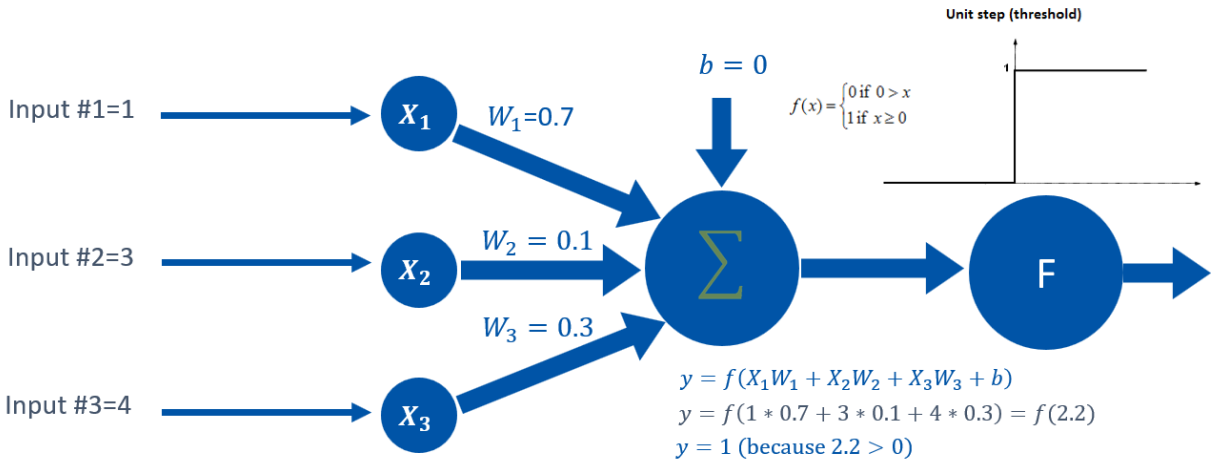


- Crédito de la foto: https://en.wikipedia.org/wiki/File:Neuron-no_labels2.png
- Crédito de la foto: https://www.flickr.com/photos/alansimpsonme/34752491090

MODELO MATEMÁTICO DE LA NEURONA: EJEMPLO

- El sesgo (bias) permite cambiar la curva de la función de activación hacia arriba o hacia abajo.
- Número de parámetros ajustables = 4 (3 pesos y 1 sesgo).
- Función de activación “F”.

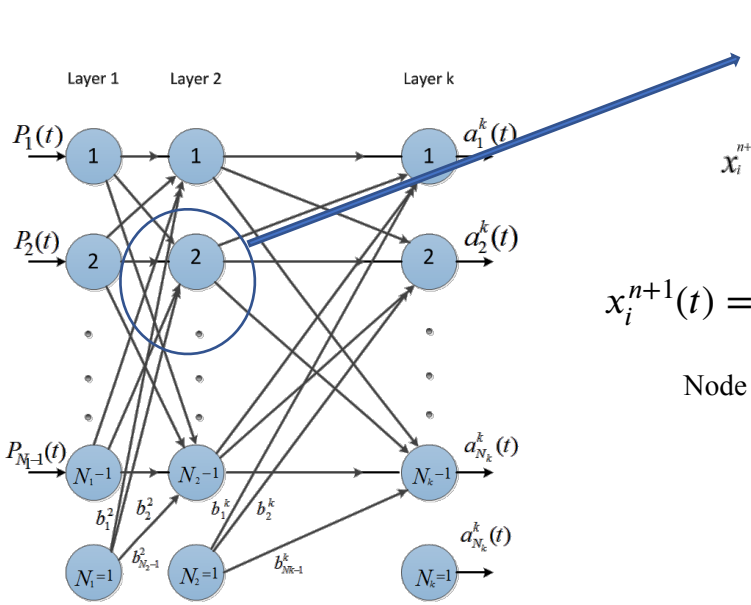


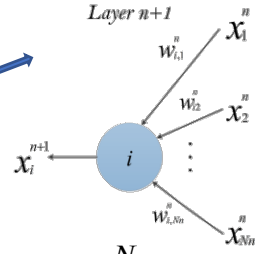


RED PERCEPTRON DE MÚLTIPLES CAPAS

- Conectemos varias de estas neuronas de forma multicapa.
- Cuantas más capas ocultas, más "profunda" se volverá la red.

$$P = \begin{bmatrix} P_1 \\ P_2 \\ ? \\ P_{N_1} \end{bmatrix}$$





$$x_i^{n+1}(t) = \varphi\left(\sum_{j=1}^{N_n} w_{i,j}^n x_j^n(t)\right)$$

Node *(n+1, i)* representation

$$\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1,\,N_1} \\ W_{21} & W_{22} & & W_{2,\,N_1} \\ \vdots & & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & & W_{m-1,\,N_1} \\ W_{m,1} & W_{m,2} & \cdots & W_{m,\,N_1} \end{bmatrix}$$

Non-Linear Sigmoid Activation function

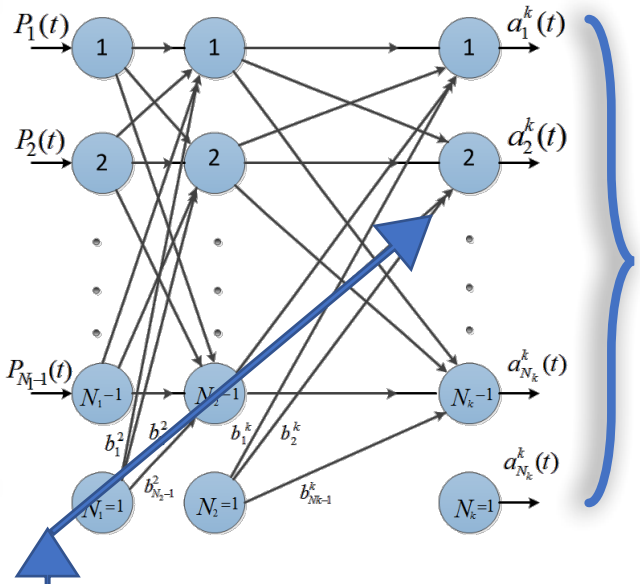
$$\varphi(w) = \frac{1}{1 + e^{-w}}$$

m: número de neuronas en la capa oculta

N_1*: número de entradas*

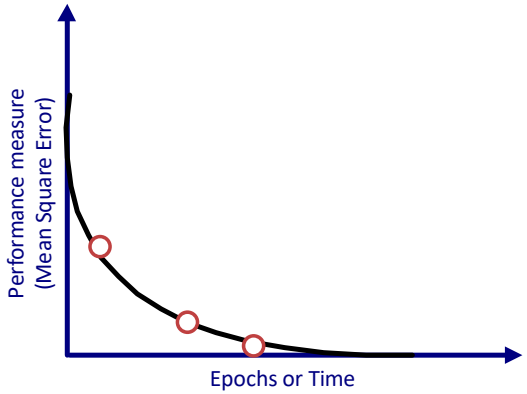
PROCESOS DE ENTRENAMIENTO Y TESTING DE RNA

Entradas de Entrenamiento X



Predicción de Salida $\hat{Y}$

Salida Deseada (Verdadera) Y

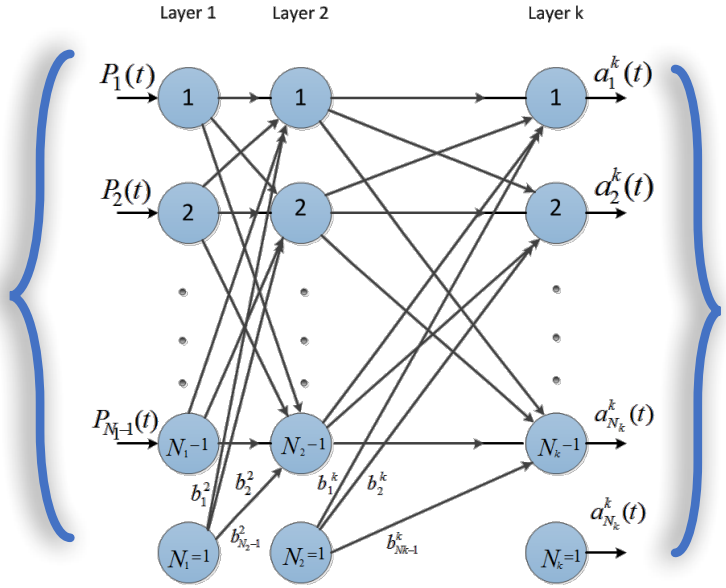


Error = $\hat{Y} - Y$

Actualizar los pesos de la red

CONGELAMOS LOS PESOS DE LA RED Y PROBAMOS LA RED CON NUEVOS DATOS QUE EL MODELO NUNCA HA VISTO ANTES

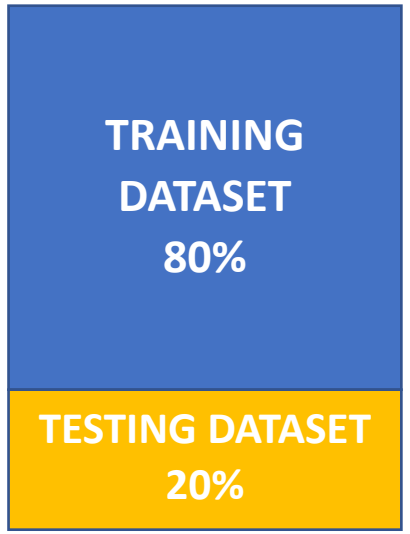
Entradas de Testing X

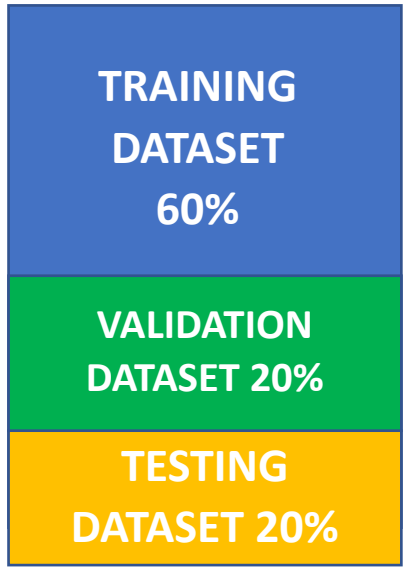


Predicción de Salida $\hat{Y}$

DIVISIÓN DE DATOS EN TRAINING Y TESTING

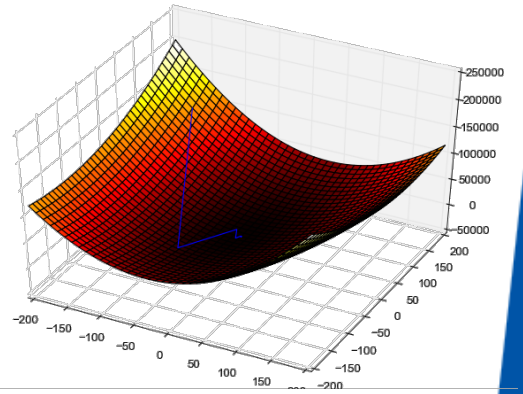
- El conjunto de datos generalmente se divide en 80% para entrenamiento y 20% para pruebas.
- A veces, también podemos incluir un conjunto de datos de validación cruzada y luego lo dividimos en segmentos de 60%, 20%, 20% para entrenamiento, validación y prueba, respectivamente (los números pueden variar).
  - Conjunto de entrenamiento: se utiliza para calcular el gradiente y actualizar los pesos de la red.
  - Conjunto de validación:
    - utilizado para la validación cruzada para evaluar la calidad del entrenamiento a medida que avanza el entrenamiento.
    - La validación cruzada se implementa para superar el ajuste excesivo que se produce cuando el algoritmo se centra en los detalles del conjunto de entrenamiento a costa de perder la capacidad de generalización.
  - Conjunto de prueba: utilizado para probar la red entrenada.

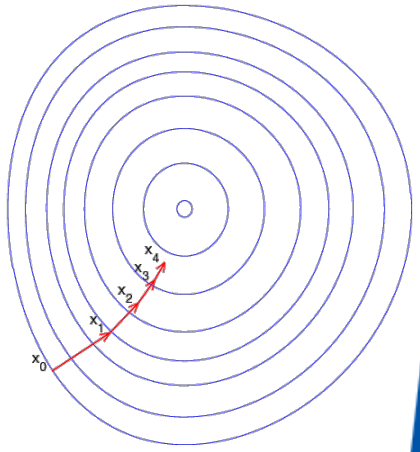


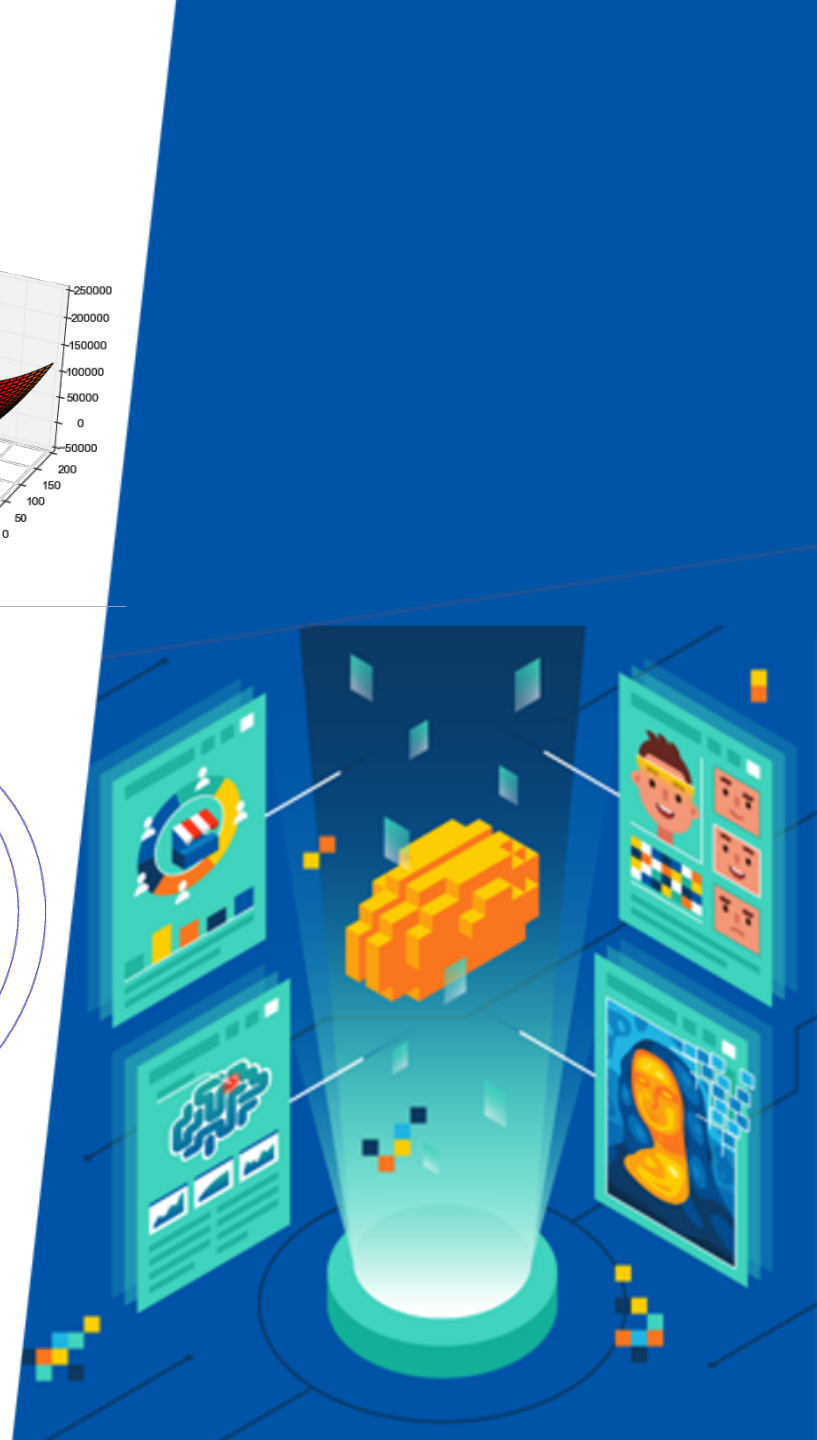


GRADIENTE DESCENDENTE

- El gradiente descendente es un algoritmo de optimización que se utiliza para obtener el peso de red optimizado y los valores del sesgo.
- Funciona intentando minimizar de forma iterativa la función de coste
- Funciona calculando el gradiente de la función de costes y moviéndose en la dirección negativa hasta que se alcanza el mínimo local / global
- Si se toma el valor positivo del gradiente, se alcanza el máximo local / global
- El tamaño de los pasos dados a cada iteración se llama tasa de aprendizaje.
- Si la tasa de aprendizaje aumenta, el área cubierta en el espacio de búsqueda aumentará para que podamos alcanzar el mínimo global más rápido.
- Sin embargo, podemos sobrepasar el objetivo
- Para tasas de aprendizaje pequeñas, el entrenamiento llevará mucho más tiempo para alcanzar valores de peso optimizados.







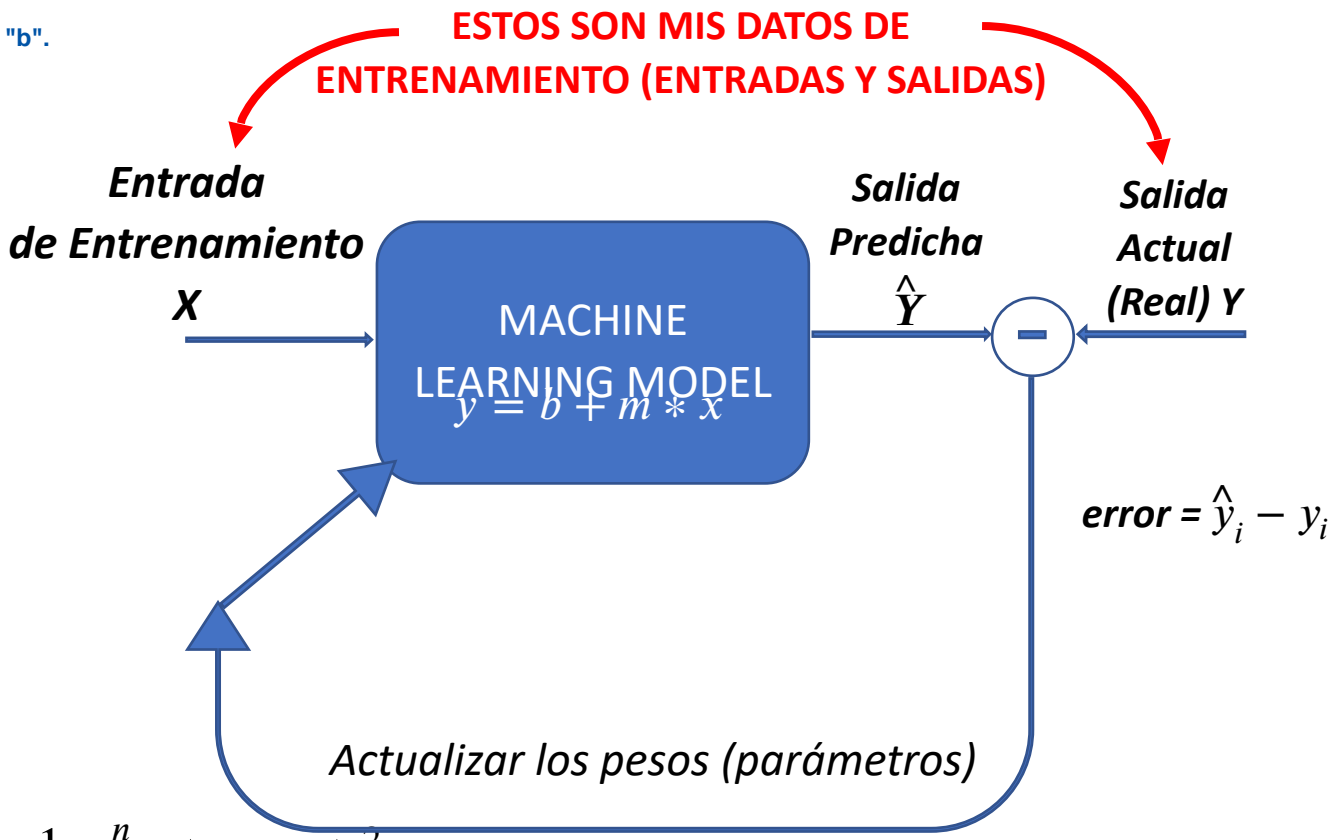
Crédito de la foto: https://commons.wikimedia.org/wiki/File:Gradient_descent_method.png
Crédito de la foto: https://commons.wikimedia.org/wiki/File:Gradient_descent.png

GRADIENTE DESCENDENTE

- Supongamos que queremos obtener los valores óptimos para los parámetros "m" y "b".

$$y = b + m * x$$

EL OBJETIVO ES ENCONTRAR LOS MEJORES PARÁMETROS



- **Primero debemos formular una función de pérdida de la siguiente manera:**

$$Cost\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(error)^2 = \frac{1}{N}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2$$

GRADIENTE DESCENDENTE

$$Loss\ Function\ f(m,b) = \frac{1}{N}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2$$

PASOS DEL GRADIENTE DESCENDENTE:

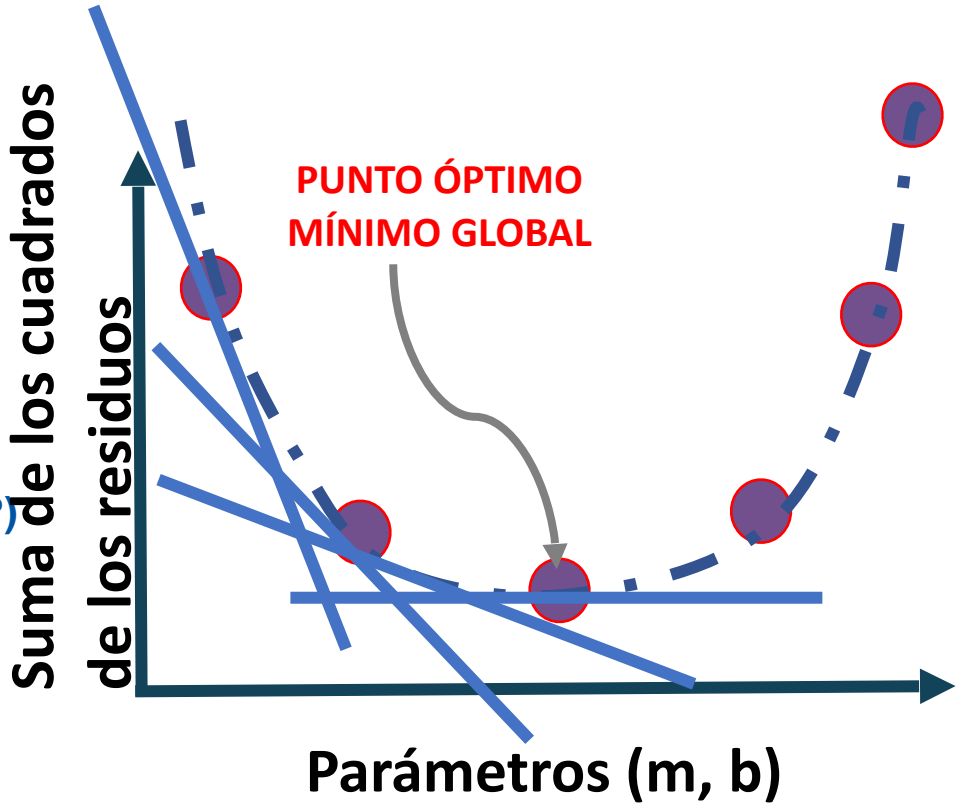
1. Se calcula el gradiente (derivada) de la función de pérdidas $\frac{\partial loss}{\partial w}$
2. Seleccionamos valores aleatorios para los pesos (m,b) y se sustituyen
3. Calculamos el tamaño del Paso (cuánto se van a actualizar los parámetros?)

$$Step\ size\ = learning\ rate * gradient = \alpha * \frac{\partial loss}{\partial w}$$

4. Actualizamos los parámetros y repetimos

$$new\ weight\ =\ old\ weight\ - step\ size$$

$$w_{new} = w_{old} - \alpha * \frac{\partial loss}{\partial w}$$



** Nota: en realidad, este gráfico es 3D y tiene tres ejes, uno para m, b y suma de residuos al cuadrado*

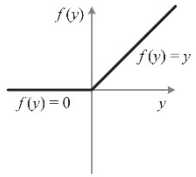
REDES NEURALES CONVOLUCIONALES: DESCRIPCIÓN GENERAL DE LA RED

CONVOLUCIÓN

KERNELS/ DETECTORES DE CARACTERÍSTICAS

CAPA CONVOLUCIONAL

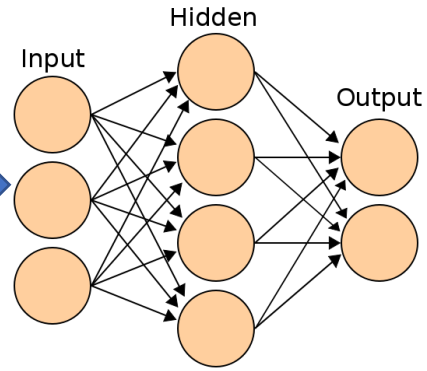
POOLING

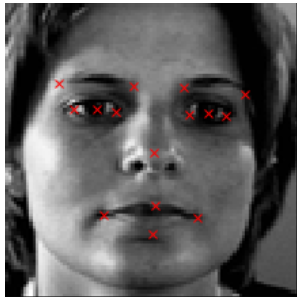


POOLING FILTERS

CAPA DE POOLING (DOWNSAMPLING)

FLATTENING





X_1
Y_1
X_2
Y_2
X_3
Y_3

PREDICCIÓN PUNTOS FACIALES CLAVE

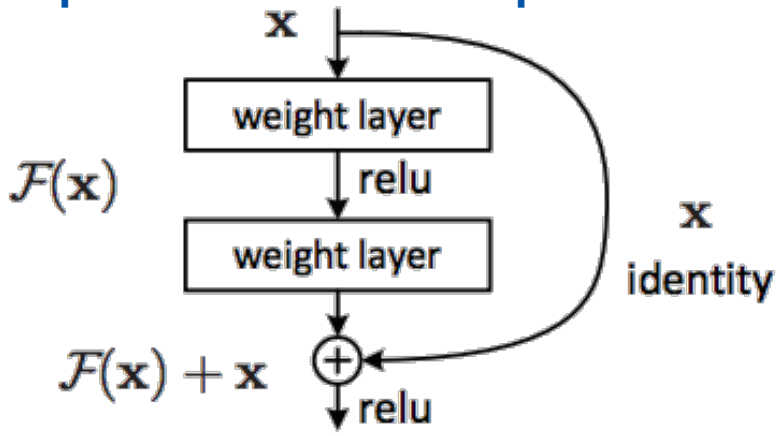
IRA
FELICIDAD
TRISTEZA
...

PREDICCIÓN EMOCIÓN

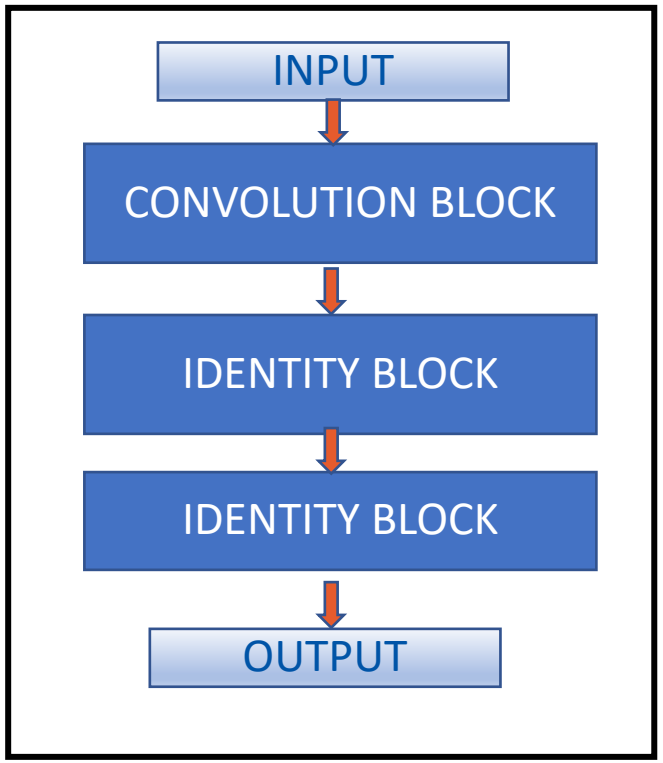
Crédito de la foto: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

RESNET (RESIDUAL NETWORK)

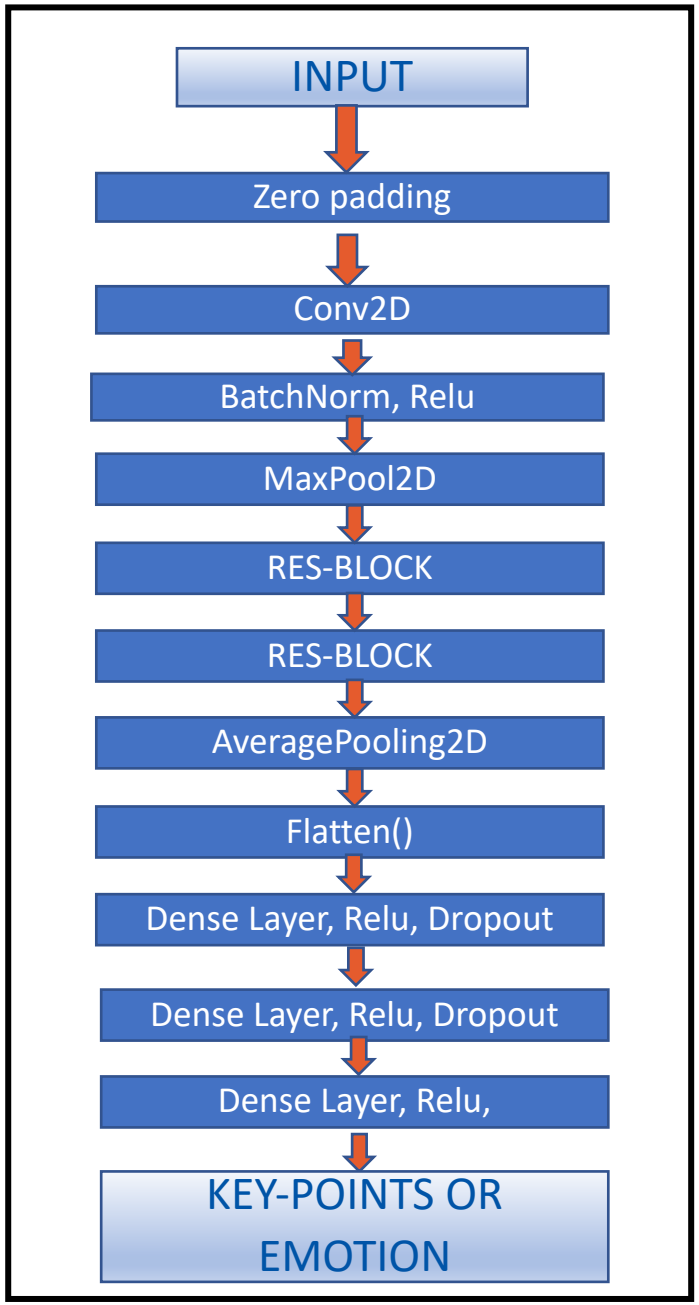
- A medida que las RNC se hacen más profundas, tienden a ocurrir el desvanecimiento del gradiente que impacta negativamente en el rendimiento de la red.
- El problema del desvanecimiento del gradiente ocurre cuando el gradiente se propaga hacia atrás a capas anteriores, lo que da como resultado un gradiente muy pequeño.
- La red neuronal residual incluye la función de "omisión de conexión" que permite el entrenamiento de 152 capas sin el problema del desvanecimiento del gradiente.
- ResNet funciona agregando "asignaciones de identidad" en la parte superior de RNC.
- ImageNet contiene 11 millones de imágenes y 11.000 categorías.
- ImageNet se utiliza para entrenar la red profunda ResNet.



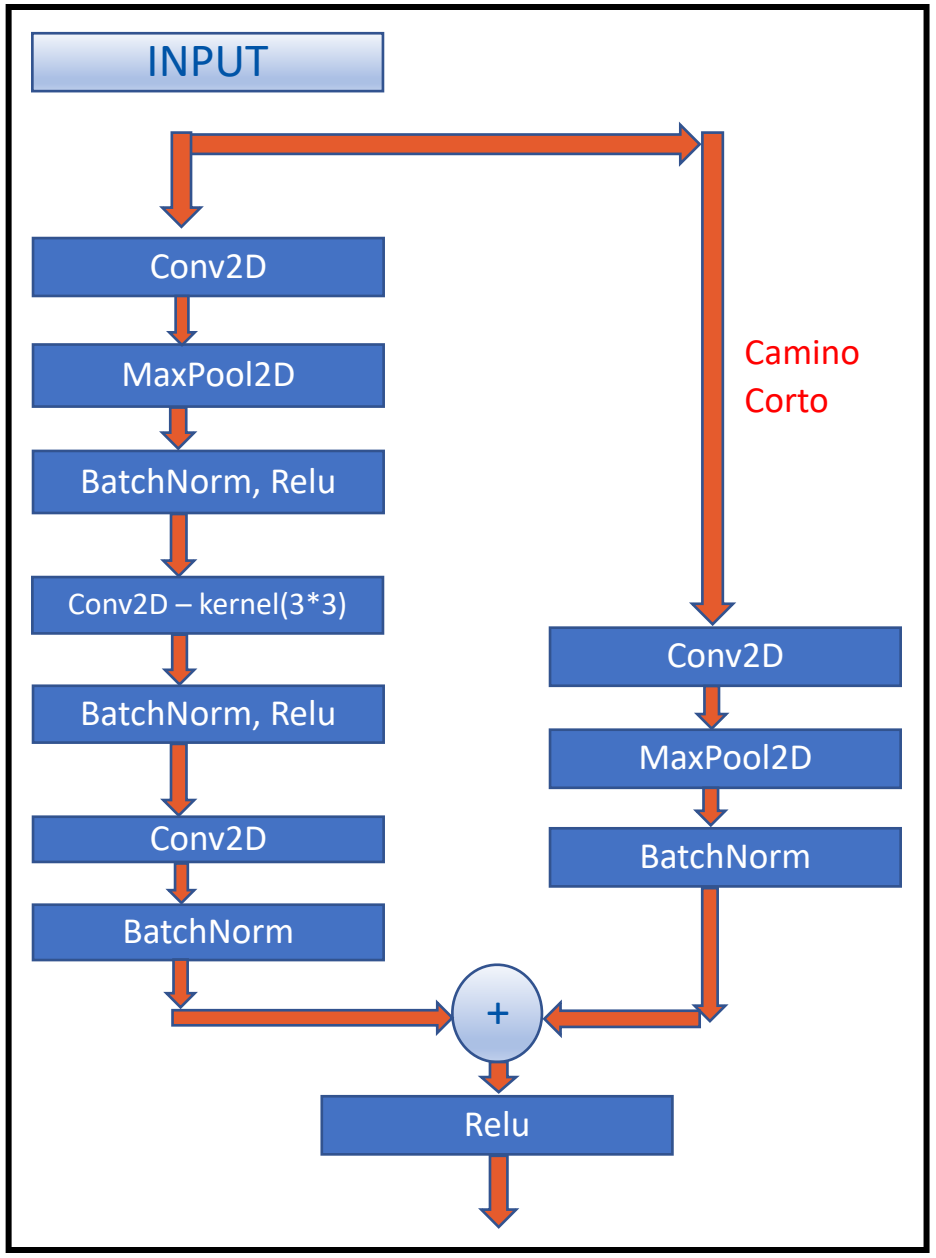
BLOQUE RES



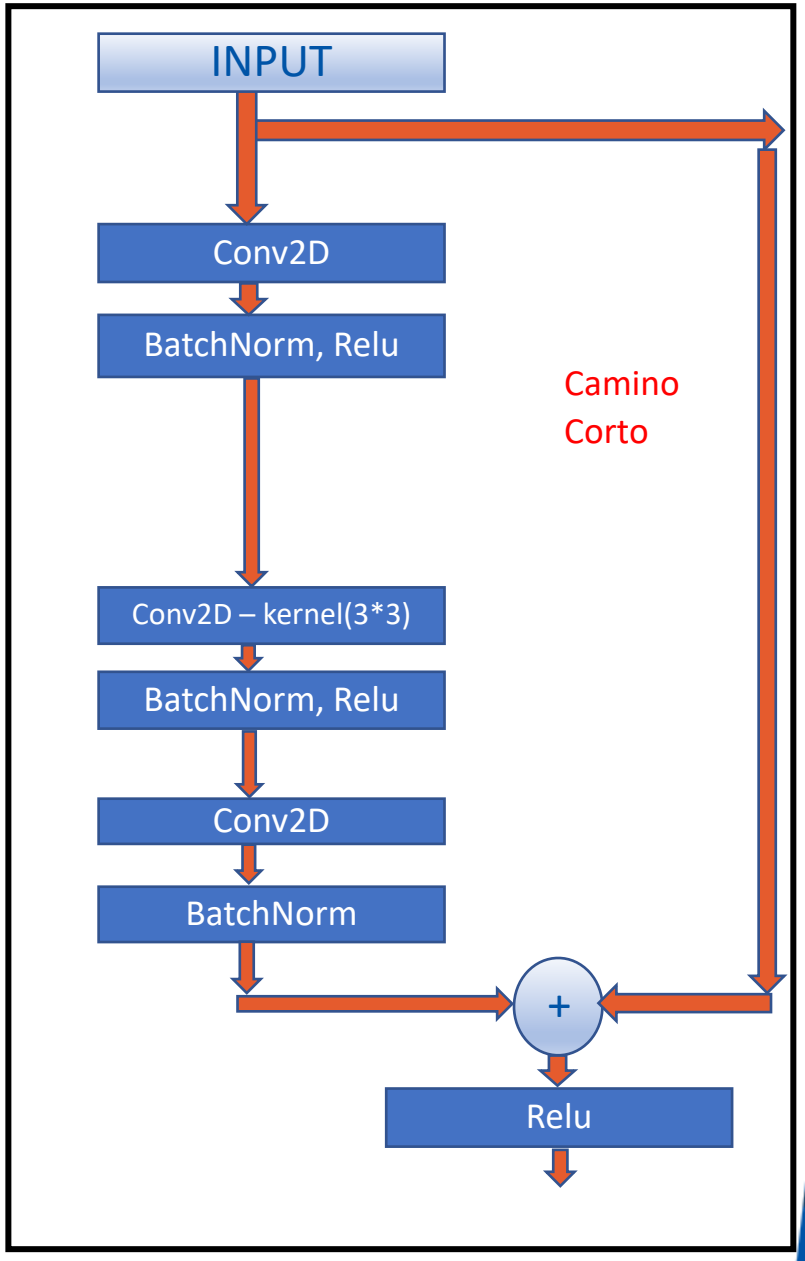
MODELO FINAL



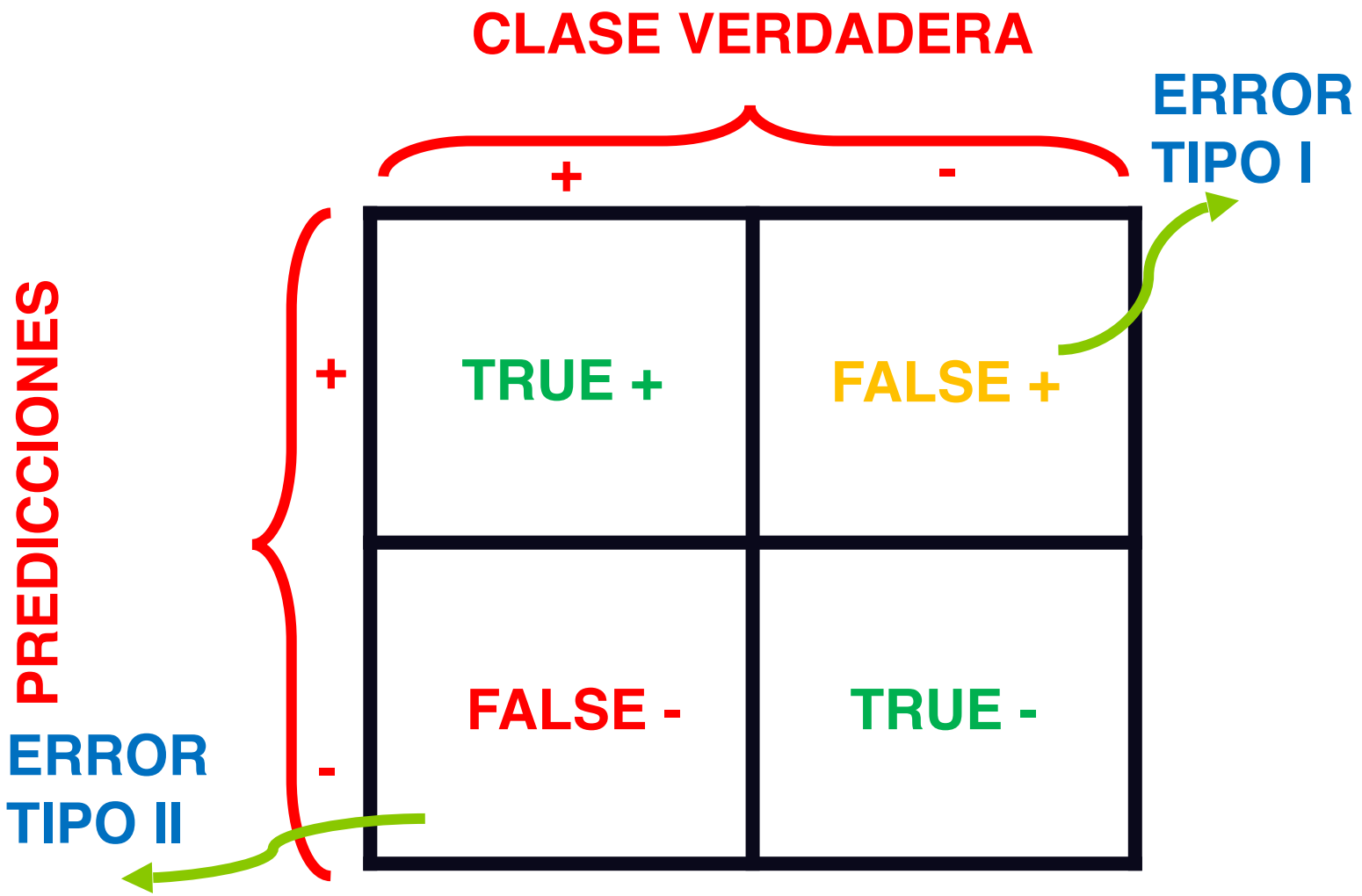
BLOQUE DE CONVOLUCIÓN



BLOQUE DE IDENTIDAD



MATRIZ DE CONFUSIÓN



DEFINICIONES Y KPIS

- Se usa una matriz de confusión para describir el desempeño de un modelo de clasificación:
  - Verdaderos positivos (TP): casos en los que el clasificador predice VERDADERO (tienen la enfermedad) y la clase correcta es VERDADERA (el paciente tiene la enfermedad).
  - Verdadero Negativo (TN): casos en los que el modelo predice FALSO (sin enfermedad) y la clase correcta es FALSO (el paciente no tiene enfermedad).
  - Falsos positivos (FP) (error de tipo I): el clasificador predice VERDADERO, pero la clase correcta es FALSO (el paciente no tiene enfermedad).
  - Falsos negativos (FN) (error de tipo II): el clasificador predice FALSO (el paciente no tiene la enfermedad), pero en realidad sí la tiene
  - Tasa de acierto de clasificación = (TP + TN) / (TP + TN + FP + FN)
  - Tasa de clasificación errónea (Tasa de error) = (FP + FN) / (TP + TN + FP + FN)
  - Precisión = TP / Total TRUE Predicciones = TP / (TP + FP) (Cuando el modelo predice la clase TRUE, ¿con qué frecuencia es correcto?)
  - Recall = TP / Actual VERDADERO = TP / (TP + FN) (cuando la clase es realmente VERDADERA, ¿con qué frecuencia el clasificador lo hace bien?

PRECISION Vs. RECALL EJEMPLO

HECHO:
100 PACIENTES EN TOTAL
91 PACIENTES SANOS
9 PACIENTES CON CANCER

CLASE VERDADERA

PREDICCIONES

	+	-
+	TP = 1	FP = 1
-	FN = 8	TN = 90

- El acierto es generalmente engañosa y no es suficiente para evaluar el desempeño de un clasificador.
  - El recall es un KPI importante en situaciones en las que:
  - El conjunto de datos está muy desbalanceado; casos en los que tienes pequeños pacientes con cáncer en comparación con los sanos.

- Acierto de clasificación = (TP+TN) / (TP + TN + FP + FN) = 91%
- Precisión = TP/Total Predicciones TRUE = TP/ (TP+FP) = ½=50%
- Recall = TP/ TRUE Reales = TP/ (TP+FN) = 1/9 = 11%

DESPLIEGUE DEL MODELO UTILIZANDO TENSORFLOW SERVING:

- Supongamos que ya entrenamos nuestro modelo y está generando buenos resultados en los datos de prueba.
- Ahora, queremos integrar nuestro modelo de Tensorflow entrenado en una aplicación web e implementar el modelo en un entorno de nivel de producción.
- El siguiente objetivo se puede obtener utilizando TensorFlow Serving. TensorFlow Serving es un sistema de publicación de alto rendimiento para modelos de aprendizaje automático, diseñado para entornos de producción.
- Con la ayuda de TensorFlow Serving, podemos implementar fácilmente nuevos algoritmos para hacer predicciones.
- Para publicar el modelo entrenado con TensorFlow Serving, necesitamos guardar el modelo en el formato que sea adecuado para entregar usando TensorFlow Serving.
- El modelo tendrá un número de versión y se guardará en un directorio estructurado.
- Una vez que se guarda el modelo, ahora podemos usar TensorFlow Serving para comenzar a realizar solicitudes de inferencia utilizando una versión específica de nuestro modelo entrenado "servible".

EJECUTAR TENSORFLOW SERVING:

- **Parámetros importantes:**
  - rest_api_port: el Puerto que usaremos para las peticiones REST.
  - model_name: la URL que usaremos para las peticiones REST, se puede elegir cualquier nombre
  - model_base_path: la ruta al directorio donde hemos guardado el modelo
- **Para más información sobre peticiones REST, comprobad: https://www.codecademy.com/articles/what-is-rest**
- **REST es una reinterpretación del protocolo HTTP donde los comandos http tienen un significado semántico.**

HACER PETICIONES CON TENSORFLOW SERVING:

- Para hacer predicciones usando TensorFlow Serving, necesitamos pasar las solicitudes de inferencia (datos de nuestra imagen) como un objeto JSON.
- Luego, usamos la librería requests de Python para realizar una solicitud por POST al modelo implementado, pasando el objeto JSON que contiene las solicitudes de inferencia (datos de nuestra imagen).
- Finalmente, obtenemos la predicción de la solicitud por POST realizada al modelo implementado y luego usamos la función argmax para encontrar la clase predicha.