“RED NEURONAL”

Clasificación de emojis

Integrantes:

Daniel Sebastian Preciado Vega.

Gerardo García Valle.

Diego Gutierrez Contreras.

Asignatura: Seminario de Algoritmia.

Sección: D11.

Nombre del profesor: Jorge Ernesto López Arce Delgado

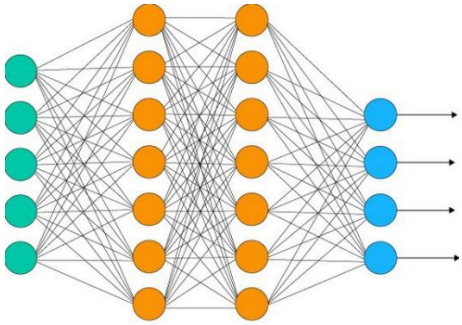
Ciclo escolar: 2023A.

Mayo 23 de 2023.

Introducción

Red Neuronal

Una red neuronal es un modelo simplificado que emula el modo en que el cerebro humano procesa la información: Funciona simultaneando un número elevado de unidades de procesamiento interconectadas que parecen versiones abstractas de neuronas.

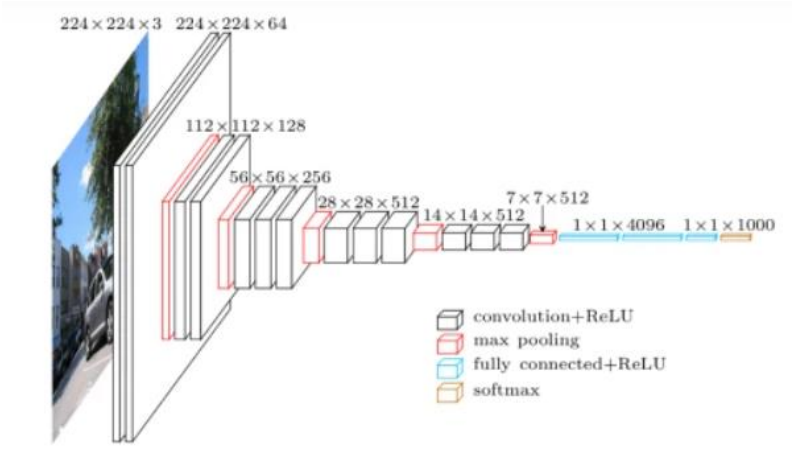


VGG

Grupo de Geometría Visual (VGG), es una arquitectura de red neuronal convolucional (CNN) profunda estándar con varias capas. Con lo profundo se refiere a que está compuesto por cierto número de capas como VGG-16 o VGG-19 que son 16 y 19 capas convolucionales.

La arquitectura fue diseñada para aumentar la profundidad de las CNN con el propósito de incrementar el rendimiento del modelo. Fue tal el éxito del aprendizaje profundo que ha influido en una amplia gama de tareas de visión artificial, donde el campo de aprendizaje automático ha estado creciendo muy rápidamente. Gracias a ello han nacido nuevas aplicaciones como el reconocimiento de imágenes, visión por computadora, reconocimiento de voz, traducción automática, imágenes médicas, robótica, entre otros.

Además, la arquitectura es la base de modelos innovadores sobre reconocimiento de objetos, por lo que es una de las más populares en el reconocimiento de imágenes.



VGG16

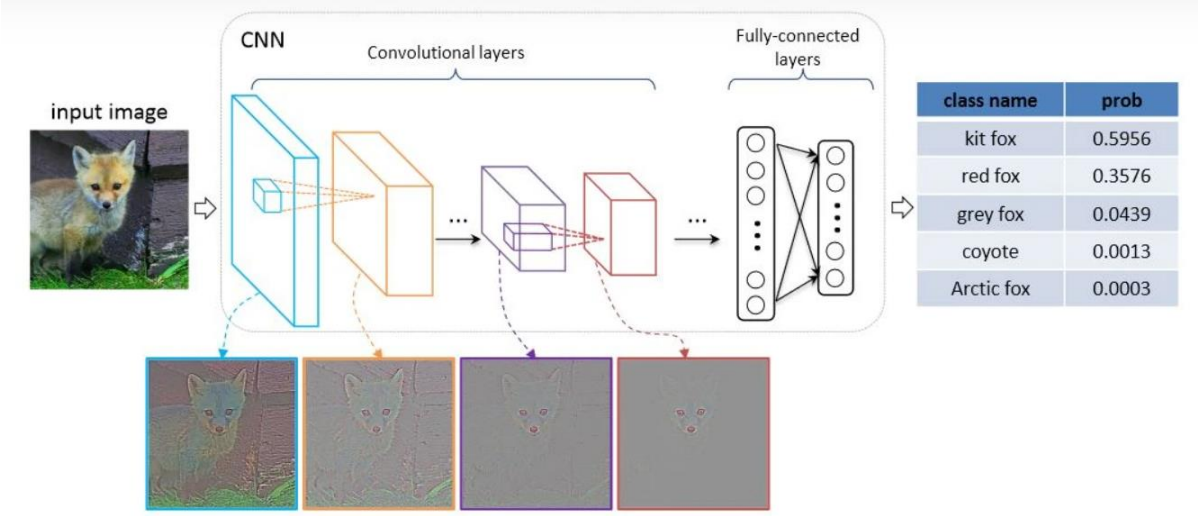
Este modelo de red neuronal convolusional compuesto por 16 capas fue propuesto por A. Zisserman y K. Simonyan que pertenecían a la universidad de Oxford. Publicaron su modelo en el artículo de investigación titulado: “Redes convolucionales muy profundas para el reconocimiento de imágenes a gran escala”.

Este modelo logra un 92,7% de precisión en las pruebas de los cinco primeros en ImageNet. Esta es un conjunto de datos que está compuesto por más de 14 millones de imágenes pertenecientes a casi 1000 clases. Por lo que fue uno de los modelos más populares ya que remplaza los filtros de gran tamaño del kernel con varios filtros del tamaño del kernel 3x3 uno tras otro. El modelo VGG16 se estrenó con GPU Nvidia Titan Black durante un tiempo.

En resumen, el modelo de 16 capas puede clasificar imágenes en 1000 categorías diferentes de objetos. Y, además, el modelo requiere que la imagen de entrada será de 224 por 224.

Arquitectura VGG

Las VGGNets están basadas en las características más esenciales de las redes neuronales convolucionales (CNN).



La VGGNet está hecha con filtros convolucionales muy pequeños. La VGG16 está compuesta por 13 capas convolucionales y tres capas completamente conectadas.

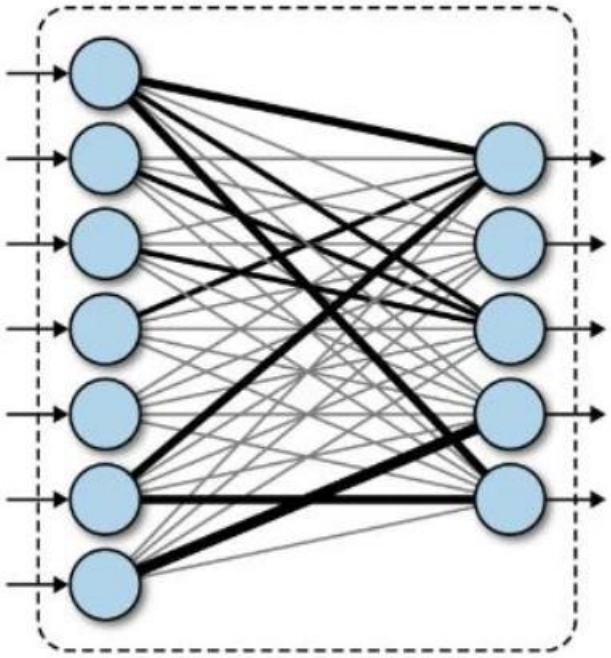
Entrada: admite un tamaño de entrada de imagen de 244x244, los creadores del modelo recortaron el parche central de 224x224 en cada imagen para mantener constante del tamaño de entrada de la imagen.

Capas convolucionales: las capas convolucionales aprovechan un campo receptivo mínimo, es decir, 3x3, el tamaño más pequeño posible que todavía captura arriba/abajo e izquierda/derecha. Además, también hay filtros de convolución 1x1 que actúan como una transformación lineal de la entrada. A esto le sigue la unidad ReLU, que es una gran innovación de AlexNet que reduce el tiempo de capacitación. ReLU significa función de activación de unidad lineal rectificada; es una función lineal por partes que generará la entrada si es positiva; de lo contrario, la salida es cero. La zancada de convulsión se fija en 1 píxel para mantener la resolución espacial preservada después de la convolución (la zancada es el número de cambios de píxel sobre la matriz de entrada).

Capas ocultas: todas las capas que están ocultas en la red VGG usan ReLU. VGG no suele aprovechar a normalización de respuesta local (LRN), ya que aumenta el

consumo de memoria y el tiempo de entrenamiento. Y, además, no mejora la precisión general.

Capas totalmente conectadas: VGGNet tiene tres capas totalmente conectadas. De esas tres, las dos primeras tienen 4096 canales cada una y la tercera tiene 1000 canales, 1 ara cada clase.



Objetivos

General

Implementar una NN (Red Neuronal) que clasifique cierto tipo de patrones en imágenes.

Particular

Objetivo particular: Entrenar una red neuronal capaz de clasificar emojis en imágenes con alta precisión utilizando técnicas de aprendizaje profundo.

Desarrollo

Implementación del modelo en código

```
from keras.models import Model
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Input
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import GlobalMaxPooling2D
from keras.layers import GlobalAveragePooling2D
from keras.preprocessing import image
from keras.utils import load_img, img_to_array
import keras.utils as image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
from keras import backend as K
from keras.applications.imagenet_utils import decode_predictions
from keras.applications.imagenet_utils import preprocess_input
#from keras.engine.topology import get_source_inputs
#from keras.applications.imagenet_utils import _obtain_input_shape # this will work for older versions of keras. 2.2.0 or before

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importación de bibliotecas: Se importan las bibliotecas necesarias para construir y entrenar la red neuronal convolucional, como Keras, NumPy, Pandas, Matplotlib y OpenCV Y Se monta Google Drive para acceder a los datos del conjunto de imágenes.

```python
def VGGupdated(input_tensor=None, classes=3):

    img_rows, img_cols = 224, 224
    img_channels = 3

    img_dim = (img_rows, img_cols, img_channels)

    img_input = Input(shape=img_dim)

    # Bloque 1
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(img_input)
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

    # Bloque 2
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

    # Bloque 3
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

    # Bloque 4
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

    # Bloque 5
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)


    # Bloque de Classificacion
    x = Flatten(name='flatten')(x)
    x = Dense(4096, activation='relu', name='fc1')(x)
    x = Dense(4096, activation='relu', name='fc2')(x)
    x = Dense(classes, activation='softmax', name='predictions')(x)


    # Crear el modelo.

    model = Model(inputs = img_input, outputs = x, name='VGGdemo')


    return model
```

Definición del modelo VGG: Se define la arquitectura del modelo VGG mediante la función VGGupdated. El modelo consta de múltiples capas convolucionales y capas totalmente conectadas.

Compilación del modelo: Se compila el modelo especificando el optimizador, la función de pérdida y las métricas a utilizar durante el entrenamiento.

```
[ ]  model = VGGupdated(classes = 3) # Una calse por emoji

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

Carga de imágenes: Se cargan las imágenes del conjunto de datos y se almacenan en un DataFrame de Pandas. También se realiza un conteo de las imágenes por categoría.

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import os

     ruta_carpeta = '/content/drive/MyDrive/emojis_dataset'

     dataset_path = os.listdir(ruta_carpeta) # emojis_dataset antes "rooms_dataset"

     tipos_emoji = os.listdir(ruta_carpeta) # tipos_emoji antes "room_types"
     #tipos_emoji.sort()
     print (tipos_emoji)  #Que emojis componen el dataset

     print("Tipos de emoji encontrados: ", len(dataset_path))
```

```
['3_Enojo', '1_Felicidad', '2_Tristeza']
Tipos de emoji encontrados:  3
```

Preprocesamiento de imágenes: Se redimensionan las imágenes y se normalizan dividiendo los valores de píxel por 255.

```
[ ]  imagenes = np.array(imagenes)

     imagenes = imagenes.astype('float32') / 255.0
     imagenes.shape
```

```
(308, 224, 224, 3)
```

Codificación de etiquetas: Se realiza una codificación de etiquetas utilizando LabelEncoder y OneHotEncoder para convertir las etiquetas de texto en vectores codificados.

```
[ ]  from sklearn.preprocessing import LabelEncoder , OneHotEncoder

     y=emojis_df['Emoji'].values

     y_labelencoder = LabelEncoder()
     y = y_labelencoder.fit_transform(y)
     print(y)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
[ ]  from sklearn.compose import ColumnTransformer
     y=y.reshape(-1,1)

     columnTransformer = ColumnTransformer([('emoji', OneHotEncoder(), [0])], remainder='passthrough')
     Y=np.array(columnTransformer.fit_transform(y))
     Y.shape  #(308, 3)
```

```
(308, 3)
```

División de datos de entrenamiento y prueba: Se dividen los datos en conjuntos de entrenamiento y prueba utilizando la función train_test_split de scikit-learn.

```
[ ]  from sklearn.utils import shuffle
     from sklearn.model_selection import train_test_split


     imagenes, Y = shuffle(imagenes, Y, random_state=1)

     train_x, test_x, train_y, test_y = train_test_split(imagenes, Y, test_size=0.9, random_state=650)

     #inpect the shape of the training and testing.
     print(train_x.shape)
     print(train_y.shape)
     print(test_x.shape)
     print(test_y.shape)
```

```
(30, 224, 224, 3)
(30, 3)
(278, 224, 224, 3)
(278, 3)
```

Entrenamiento del modelo: Se entrena el modelo utilizando los datos de entrenamiento.

```
[ ]  model.fit(train_x, train_y, epochs = 10, batch_size = 32)
```

```
Epoch 1/10
1/1 [==============================] - 26s 26s/step - loss: 1.0981 - accuracy: 0.4667
Epoch 2/10
1/1 [==============================] - 0s 410ms/step - loss: 34.3699 - accuracy: 0.4667
Epoch 3/10
1/1 [==============================] - 0s 406ms/step - loss: 1.1015 - accuracy: 0.3000
Epoch 4/10
1/1 [==============================] - 0s 415ms/step - loss: 1.0987 - accuracy: 0.3000
Epoch 5/10
1/1 [==============================] - 0s 402ms/step - loss: 1.0887 - accuracy: 0.4667
Epoch 6/10
1/1 [==============================] - 0s 401ms/step - loss: 1.0588 - accuracy: 0.4667
Epoch 7/10
1/1 [==============================] - 0s 412ms/step - loss: 1.0736 - accuracy: 0.4667
Epoch 8/10
1/1 [==============================] - 0s 405ms/step - loss: 1.0716 - accuracy: 0.4667
Epoch 9/10
1/1 [==============================] - 0s 400ms/step - loss: 1.0567 - accuracy: 0.4667
Epoch 10/10
1/1 [==============================] - 0s 399ms/step - loss: 1.1024 - accuracy: 0.4667
<keras.callbacks.History at 0x7f92c273bc70>
```

```
[ ]  preds = model.evaluate(test_x, test_y)
     print ("Errores = " + str(preds[0]))
     print ("Aciertos = " + str(preds[1]))
```

```
9/9 [==============================] - 9s 510ms/step - loss: 1.0598 - accuracy: 0.4029
Errores = 1.0598065853118896
Aciertos = 0.40287768840789795
```

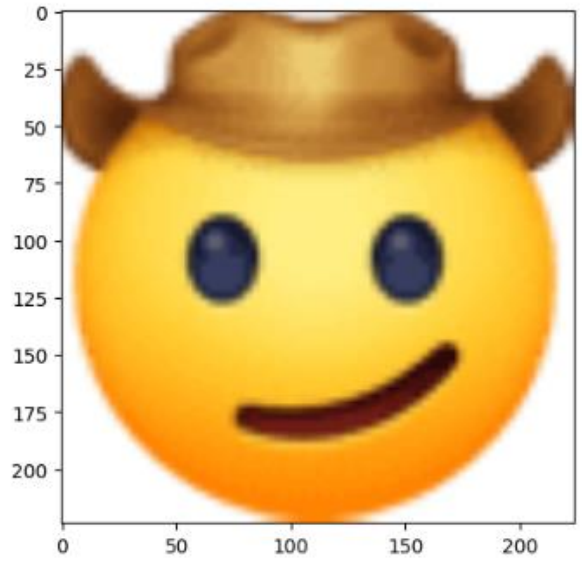
Evaluación del modelo: Se evalúa el modelo utilizando los datos de prueba y se obtienen las métricas de pérdida y precisión.

```
[ ]  from matplotlib.pyplot import imread
     from matplotlib.pyplot import imshow

     #img_path = '/content/drive/MyDrive/emojis_validar/cara_enojada.png'
     #img_path = '/content/drive/MyDrive/emojis_validar/cara_lagrima.png'
     img_path = '/content/drive/MyDrive/emojis_validar/cara_feliz.png'

     #img = image.load_img(img_path, target_size=(224, 224))
     load_img(img_path, target_size=(224, 224))

     #x = image.img_to_array(img)
     x = img_to_array(img)
     x = np.expand_dims(x, axis=0)
     x = preprocess_input(x)

     print('Input image shape:', x.shape)

     my_image = imread(img_path)
     imshow(my_image)
```

```
Input image shape: (1, 224, 224, 3)
<matplotlib.image.AxesImage at 0x7f92c08841c0>
```

Prueba con una imagen de usuario: Se carga una imagen proporcionada por el usuario, se realiza el preprocesamiento necesario y se realiza una predicción utilizando el modelo entrenado. Luego, se muestra el resultado de la predicción, que indica la probabilidad de cada categoría de emoji.

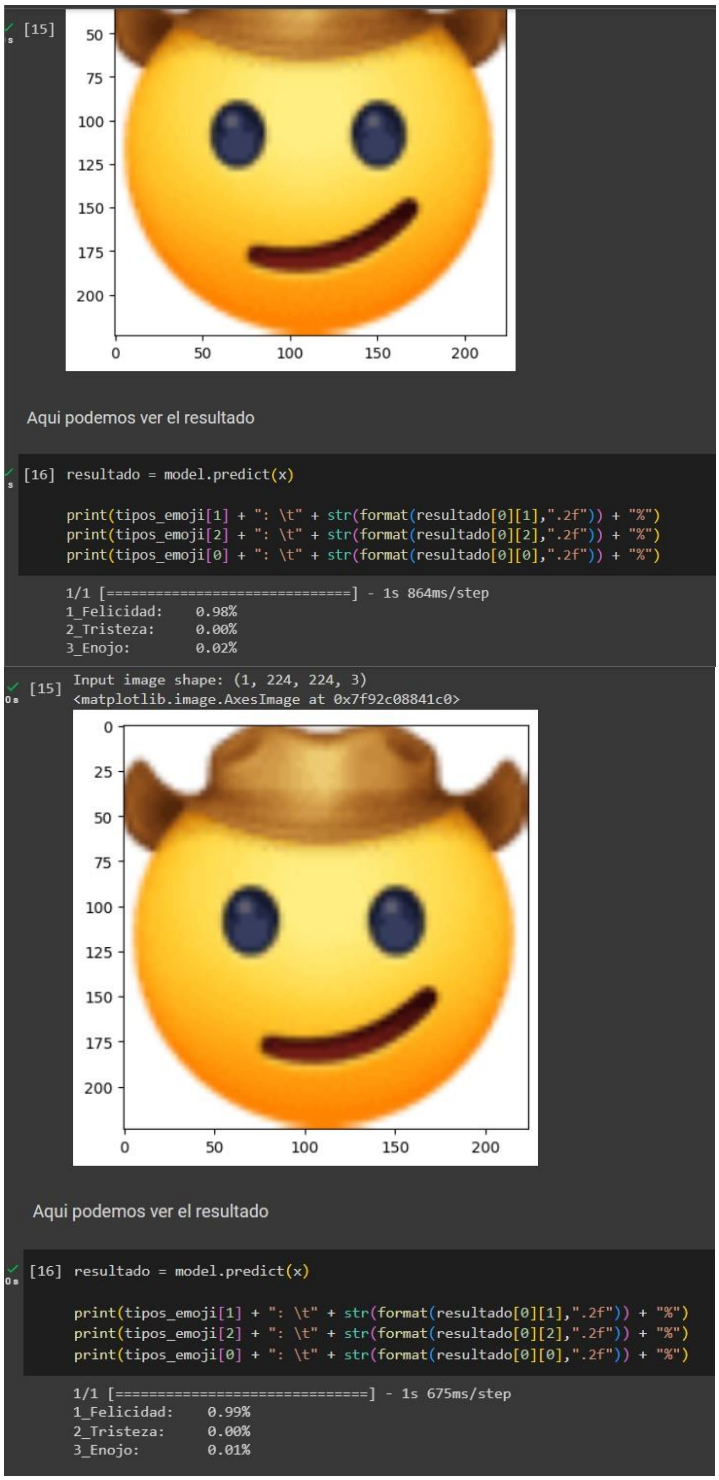
```
Input image shape: (1, 224, 224, 3)
<matplotlib.image.AxesImage at 0x7f92c08841c0>
```



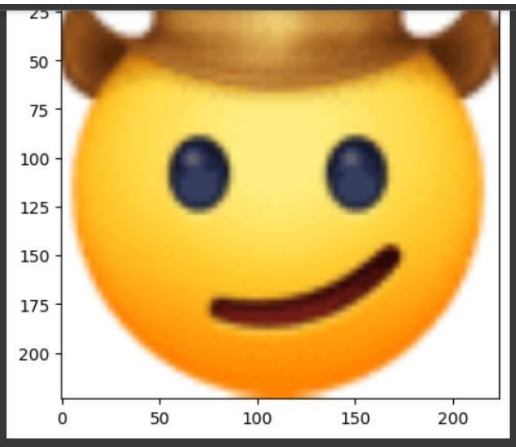
Aqui podemos ver el resultado

```
[ ]  resultado = model.predict(x)

     print(tipos_emoji[1] + ": \t" + str(format(resultado[0][1],".2f")) + "%")
     print(tipos_emoji[2] + ": \t" + str(format(resultado[0][2],".2f")) + "%")
     print(tipos_emoji[0] + ": \t" + str(format(resultado[0][0],".2f")) + "%")

     1/1 [==============================] - 1s 675ms/step
     1_Felicidad:    0.99%
     2_Tristeza:     0.00%
     3_Enojo:        0.01%
```

Resultados


[15]
Aqui podemos ver el resultado
[16] resultado = model.predict(x)
print(tipos_emoji[1] + ": \t" + str(format(resultado[0][1],".2f")) + "%")
print(tipos_emoji[2] + ": \t" + str(format(resultado[0][2],".2f")) + "%")
print(tipos_emoji[0] + ": \t" + str(format(resultado[0][0],".2f")) + "%")
1/1 [==============================] - 1s 864ms/step
1_Felicidad: 0.98%
2_Tristeza: 0.00%
3_Enojo: 0.02%
[15] Input image shape: (1, 224, 224, 3)
<matplotlib.image.AxesImage at 0x7f92c08841c0>
Aqui podemos ver el resultado
[16] resultado = model.predict(x)
print(tipos_emoji[1] + ": \t" + str(format(resultado[0][1],".2f")) + "%")
print(tipos_emoji[2] + ": \t" + str(format(resultado[0][2],".2f")) + "%")
print(tipos_emoji[0] + ": \t" + str(format(resultado[0][0],".2f")) + "%")
1/1 [==============================] - 1s 675ms/step
1_Felicidad: 0.99%
2_Tristeza: 0.00%
3_Enojo: 0.01%

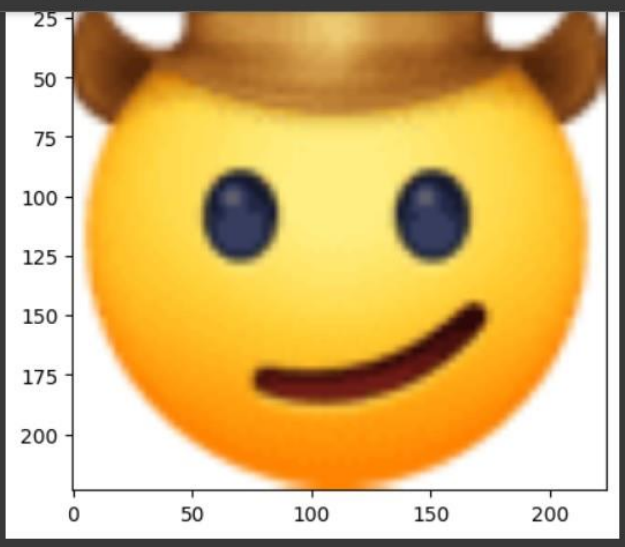
[15]



Aqui podemos ver el resultado

```
[16] resultado = model.predict(x)

     print(tipos_emoji[1] + ": \t" + str(format(resultado[0][1],".2f")) + "%")
     print(tipos_emoji[2] + ": \t" + str(format(resultado[0][2],".2f")) + "%")
     print(tipos_emoji[0] + ": \t" + str(format(resultado[0][0],".2f")) + "%")
```

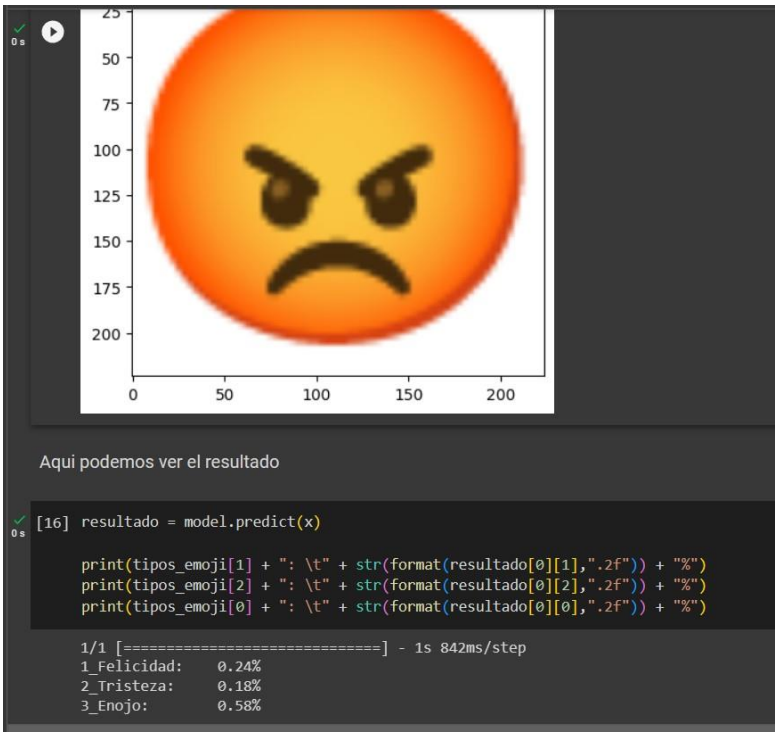
```
1/1 [==============================] - 1s 661ms/step
1_Felicidad:    0.60%
2_Tristeza:     0.17%
3_Enojo:        0.23%
```



Aqui podemos ver el resultado

```
[51] resultado = model.predict(x)

     print(tipos_emoji[1] + ": \t" + str(format(resultado[0][1],".2f")) + "%")
     print(tipos_emoji[2] + ": \t" + str(format(resultado[0][2],".2f")) + "%")
     print(tipos_emoji[0] + ": \t" + str(format(resultado[0][0],".2f")) + "%")
```

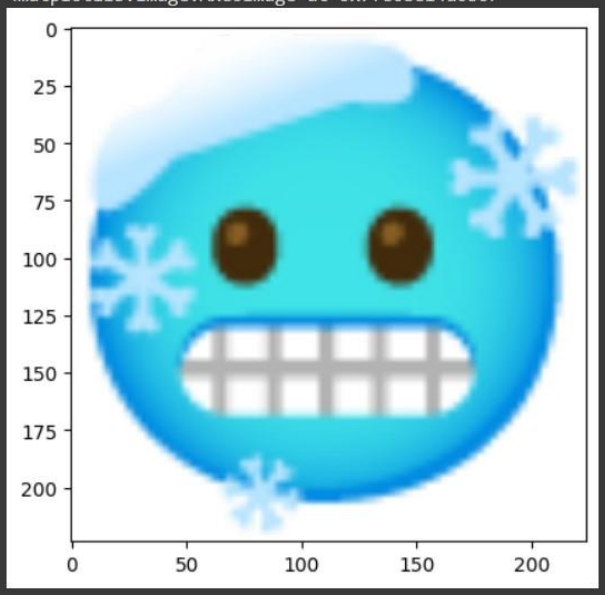
```
1/1 [==============================] - 0s 30ms/step
1_Felicidad:    0.50%
2_Tristeza:     0.12%
3_Enojo:        0.38%
```



Aqui podemos ver el resultado

```
[16] resultado = model.predict(x)

     print(tipos_emoji[1] + ": \t" + str(format(resultado[0][1],".2f")) + "%")
     print(tipos_emoji[2] + ": \t" + str(format(resultado[0][2],".2f")) + "%")
     print(tipos_emoji[0] + ": \t" + str(format(resultado[0][0],".2f")) + "%")

     1/1 [==============================] - 1s 842ms/step
     1_Felicidad:    0.24%
     2_Tristeza:     0.18%
     3_Enojo:        0.58%
```

```
[ ]  Input image shape: (1, 224, 224, 3)
     <matplotlib.image.AxesImage at 0x7fe68e14a0b0>
```



Aqui podemos ver el resultado

```
[ ]  print(model.predict(x))

     1/1 [==============================] - 0s 151ms/step
     [[0.26378742 0.00850928 0.19393578 0.40493423 0.12883334]]
```

Conclusiones

La Red Neuronal Convolucional es un modelo inspirado en el cerebro humano que utiliza unidades de procesamiento interconectadas para procesar información. Se compone de capas convolucionales y capas totalmente conectadas y se utiliza en tareas de visión artificial y reconocimiento de imágenes.

La arquitectura VGG (Grupo de Geometría Visual) es una CNN profunda estándar que aumenta la profundidad de las redes neuronales convolucionales para mejorar el rendimiento del modelo. Ha sido ampliamente utilizada en diversas aplicaciones, como reconocimiento de objetos, visión por computadora e imágenes médicas.

El código proporcionado implementa la arquitectura VGG utilizando Keras. Incluye la importación de bibliotecas necesarias, la definición del modelo VGG, la compilación del modelo, la carga y preprocesamiento de imágenes, la codificación de etiquetas, la división de datos en conjuntos de entrenamiento y prueba, el entrenamiento y evaluación del modelo, y una sección para realizar pruebas con una imagen de usuario.

Referencias

- Gaudenz Bosch. (S/F). Redes convolucionales muy profundas VGG (VGGNet): lo que necesita saber. Recuperado de https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

- Boesch, G. (2023, March 16). *VGG Very Deep Convolutional Networks (VGGNet) – What you need to know*. viso.ai. https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

- Nepal, P. (2021, December 15). VGGNet Architecture Explained - Analytics Vidhya - Medium. Medium. https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6

- Baheti, P. (2023, April 24). A Comprehensive Guide to Convolutional Neural Networks. *V7*. https://www.v7labs.com/blog/convolutional-neural-networks-guide#h2