CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OPTATIVA PROFESIONALIZANTE II: MACHINE LEARNING Y DEEP LEARNING

10° "A"

PROYECTO FINAL

Profesor: Dr. Francisco Javier Luna Rosas

Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags.**, 15** de mayo de 2023

Proyecto Final

Introducción

Un sistema de reconocimiento de patrones de rostros permite identificar a una persona en una imagen digital, mediante un análisis de las características faciales del sujeto extraídas y comparándolas en una base de datos. Las expresiones faciales son los cambios faciales como consecuencia de los estados emocionales internos o las comunicaciones sociales de una persona y este es un tema investigado activamente en la visión por computadora (Tian et al. 2011).

En este proyecto se desarrollará una Red Neuronal Convolucional (CNN), la cual podrá clasificar diferentes expresiones faciales de una persona, colocando las etiquetas de la emoción detectada: Feliz, Enojado, Triste, Sorprendido y Neutral.

El desarrollo del reconocimiento de expresiones faciales de una persona será dividido en tres secciones (Figura 1).

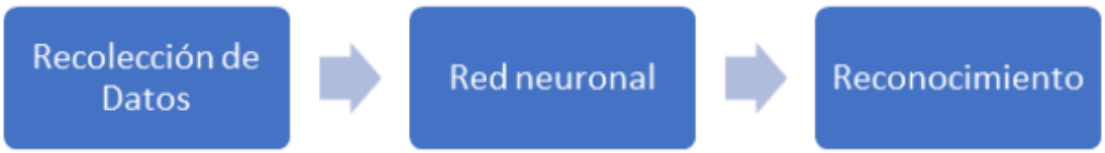


Figura 1. Gráfica de flujo de tres secciones en el desarrollo del sistema reconocedor de expresión facial.

Recolección de Datos. Se hará una recolección de las imágenes de expresiones faciales por medio de una webcam.

Red CNN. Las imágenes de expresiones faciales se usarán como Dataset para generar un Modelo de Red Neuronal CNN.

Reconocimiento de Expresiones Faciales de la CNN. Este modelo servirá para hacer el reconocimiento de la expresión facial de la imagen capturada por medio de una webcam en tiempo real.

Propósito de la actividad

Comprender el funcionamiento de las Redes Neuronales Convolucionales (CNN) aplicadas al reconocimiento de expresiones faciales en tiempo real.

Instrucciones

Analizar, implementar y probar una Red Neuronal Covolucional (CNN) para reconocer expresiones faciales en tiempo real, la implementación de la CNN será en el lenguaje de su preferencia.

Se presentará en un archivo PDF y en un archivo auto-reproducible HTML, que deberán contener:

- Portada
- Análisis de la CNN (Generación del Dataset, Generación del Modelo CNN, Reconocimiento de Expresiones Faciales en Tiempo Real).
- Implementación de la CNN (Implementación de la CNN para Reconocer Expresiones Faciales en Tiempo Real).
- Prueba de la CNN (Prueba de la CNN para Reconocer Expresiones Faciales en Tiempo Real).
- Conclusiones.
- Referencias en formato APA

Letra Arial, tamaño 12 e interlineado de 1.5 (solo archivo *.PDF).

Características y/o criterios a evaluar

Calidad de la práctica, ortografía y cumplir claramente con los temas solicitados en el apartado de instrucciones.

El Deep Learning es un área de la Inteligencia Artificial que se enfoca en el desarrollo de algoritmos de aprendizaje automático que simulan el comportamiento del cerebro humano mediante el uso de redes neuronales artificiales. Estas redes neuronales son capaces de aprender a partir de grandes cantidades de datos y realizar tareas complejas de forma autónoma, lo que ha llevado a importantes avances en campos como el procesamiento de imágenes, el reconocimiento de voz, la traducción de idiomas y muchos otros.

El reconocimiento facial es una de las aplicaciones más populares del Deep Learning, que ha encontrado su uso en diversas áreas, como la seguridad, la publicidad y la psicología. El reconocimiento facial se refiere a la capacidad de las máquinas para detectar, analizar y reconocer caras humanas a partir de imágenes o vídeos. A través del Deep Learning, es posible desarrollar modelos que sean precisos y confiables en esta tarea.

Las redes neuronales son la columna vertebral del Deep Learning. Son modelos matemáticos que simulan la estructura y el comportamiento de las neuronas biológicas en el cerebro humano. Estas redes están compuestas por capas de neuronas interconectadas y cada neurona procesa y transmite información a otras neuronas de la siguiente capa. La arquitectura y la topología de estas redes neuronales pueden variar según la tarea que se esté abordando.

En el contexto de la clasificación de emociones, los proyectos de Deep Learning se enfocan en la identificación de las emociones humanas a través del reconocimiento facial. Los modelos de Deep Learning pueden detectar y analizar características faciales sutiles, como las expresiones de los ojos, la boca, las cejas y la frente, para clasificar las emociones en categorías como la felicidad, la tristeza, el enojo y el miedo. Esto tiene aplicaciones en la psicología, la publicidad, la salud mental y muchos otros campos.

A continuación se presenta el proyecto de Deep Learning para reconocimiento facial y clasificación de emociones. El uso de redes neuronales artificiales en este proyecto permite el análisis preciso de grandes cantidades de datos de imágenes faciales para identificar las emociones humanas. Esto tiene un gran potencial en

diversos campos y podría ayudar a comprender mejor la forma en que las emociones influyen en nuestro comportamiento y bienestar.

La primera parte de este proyecto aborda la construcción del conjunto de imágenes con la que se entrenará la red neuronal; para ello se construyó el siguiente código como función para la captura de imágenes:

```
import cv2
import os

def captureImages(directory = 'img/Feliz'):
    web_cam = cv2.VideoCapture(0) # Webcam
    faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml') # Carga Haarcascades
    count = 0

    while(web_cam.isOpened()):
        ret, frame = web_cam.read() # Lee la webcam

        # Preprocesamiento
        grises = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        rostro = faceCascade.detectMultiScale(grises, 1.5, 5)
        cv2.imshow("Creando Dataset", frame) # Muestra lectura de la webcam

        for(x,y,w,h) in rostro:
            cv2.rectangle(frame, (x,y), (x + w, y + h), (255, 0, 0), 4) # Enmarca rostro
            count += 1 # Rostros detectados
            cv2.imwrite(directory + str(count) + '.jpg', grises[y:y + h, x:x + w]) # Guarda el rostro
            #

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        elif count >= 400:
            break

    # Liberamos la captura
    web_cam.release()
    cv2.destroyAllWindows()

    return
```

De este modo, se llama la función para capturar las imágenes para cada una de las cinco emociones. Se muestra la ejecución y ejemplo de la emoción etiquetada como "Feliz" a continuación:

```
captureImages("img/Feliz/")
```

Se observan algunos ejemplos de captura a continuación:

A continuación se observa la ejecución para la captura de la etiqueta "Enojado":

```
captureImages("img/Enojado/")
```

Se observan algunos ejemplos de captura a continuación:

A continuación se observa la ejecución para la captura de la etiqueta "Sorprendido":

```
captureImages("img/Sorprendido/")
```

Se observan algunos ejemplos de captura a continuación:

A continuación se observa la ejecución para la captura de la etiqueta "Triste":

```
captureImages("img/Triste/")
```

Se observan algunos ejemplos de captura a continuación:

A continuación se observa la ejecución para la captura de la etiqueta "Neutral":

```
captureImages("img/Neutral/")
```

Se observan algunos ejemplos de captura a continuación:

La fase dos de este proyecto es el entrenamiento de la red neuronal que predice los sentimientos. Para ello se realizó el siguiente código a modo de función y conseguir el entrenamiento deseado:

```
from __future__ import print_function
import os

import tensorflow.keras as keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras.optimizers import RMSprop, SGD, Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

def CNN():
    # Configuración

    # Directorio de imágenes
    train_data_dir = 'img/'
    validation_data_dir = 'img/'

    # Función de lotes con modificaciones aleatorias
    train_datagen = ImageDataGenerator(
        rotation_range = 30,
        width_shift_range = 0.4,
        height_shift_range = 0.4,
        shear_range = 0.3,
        zoom_range = 0.3,
        horizontal_flip = True,
        rescale = 1./255)
    validation_datagen = ImageDataGenerator(rescale = 1./255) # Cambiar la escala de los valores en la matriz
```

```
# Generación de lotes normalizados
batch_size = 32
img_rows, img_cols = 48, 48
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size = (img_rows, img_cols), # Cada imagen se redimensionará a este tamaño
    color_mode = 'grayscale',
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = True) # Mezclar el orden

validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size = (img_rows, img_cols),
    color_mode = 'grayscale',
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = True)

# Modelado de la CNN

num_classes = 5 # Feliz, enojado
model = Sequential()

# Block 1 (esta capa crea un núcleo de convolución que se convoluciona con la entrada de la capa para producir un tensor de
model.add(Conv2D(32, 3, padding = 'same', kernel_initializer = 'he_normal', input_shape = (img_rows, img_cols, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization()) # Entradas cambian, lo que provoca la variedad de no estacionariedad

model.add(Conv2D(32, 3, padding = 'same', kernel_initializer = 'he_normal', input_shape = (img_rows, img_cols, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization()) # Entradas cambian, lo que provoca la variedad de no estacionariedad

model.add(MaxPooling2D(pool_size = (2, 2))) # Reduce la dimensionalidad de la imagen, tomando grupos de 2x2 y nos quedamos c
model.add(Dropout(0.2)) # La capa de abandono establece aleatoriamente las unidades de entrada en 0 para evitar sobreajuste

# Block 2
model.add(Conv2D(64, 3, padding = 'same', kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())

model.add(Conv2D(64, 3, padding = 'same', kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.2))

# Block 3
model.add(Conv2D(128, 3, padding = 'same', kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
```

```
model.add(Conv2D(128, 3, padding = 'same', kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.2))

# Block 4
model.add(Conv2D(256, 3, padding = 'same', kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())

model.add(Conv2D(256, 3, padding = 'same', kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())

model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.2))

# Block 5
model.add(Flatten())
model.add(Dense(64, kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Block 6
model.add(Dense(64, kernel_initializer = 'he_normal'))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Block 7
model.add(Dense(num_classes, kernel_initializer = 'he_normal'))
model.add(Activation('softmax'))

print(model.summary()) # Resumen de cada capa

model.compile(optimizer = Adam(lr = 0.001), # Optimizer that implements the Adam algorithm
              loss = 'categorical_crossentropy', # Calcula la pérdida de entropía cruzada entre las etiquetas y predicciones
              metrics = ['accuracy']) # Calcula la frecuencia con la que las predicciones son iguales a las etiquetas

# Entrenamiento de la CNN

# Se detiene el entrenamiento cuando una métrica monitoreada haya dejado de mejorar
earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0, # min change
                          patience = 5, # no change
                          verbose = 1,
                          restore_best_weights = True)
```

```
# Configuración para guardar un modelo
checkpoint = ModelCheckpoint(filepath = 'Modelo_3.h5', # Nombre
                             monitor = 'val_loss',
                             verbose = 1,
                             save_best_only = True,
                             mode = 'min')
                             #patience = 5,
                             #verbose = 1,
                             #min_delta = 0.0001)

reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.2,
                              patience=5,
                              verbose=1,
                              min_lr=0.00001)

callbacks = [earlystop, checkpoint, reduce_lr]
epochs = 25
history = model.fit(
    train_generator, # 1,600 imágenes
    steps_per_epoch = None, # Hasta que se acaben las imágenes de train según los lotes es la época
    epochs = epochs,
    callbacks = callbacks,
    validation_data = validation_generator, # 400 imágenes
    validation_steps = None) # Hasta que se acaben las imágenes de validation según los lotes es la época

return
```

La ejecución de la función se da a continuación:

```
Found 2002 images belonging to 5 classes.
Found 2002 images belonging to 5 classes.
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 48, 48, 32)	320
activation_22 (Activation)	(None, 48, 48, 32)	0
batch_normalization_20 (BatchNormalization)	(None, 48, 48, 32)	128
conv2d_17 (Conv2D)	(None, 48, 48, 32)	9248
activation_23 (Activation)	(None, 48, 48, 32)	0
batch_normalization_21 (BatchNormalization)	(None, 48, 48, 32)	128

```
Epoch 22: val_loss improved from 0.16657 to 0.14275, saving model to Modelo_3.h5
63/63 [==============================] - 29s 469ms/step - loss: 0.4579 - accuracy: 0.8407 - val_loss: 0.1428 - val_accuracy:
0.9800 - lr: 0.0010
Epoch 23/25
63/63 [==============================] - ETA: 0s - loss: 0.4298 - accuracy: 0.8482
Epoch 23: val_loss did not improve from 0.14275
63/63 [==============================] - 28s 447ms/step - loss: 0.4298 - accuracy: 0.8482 - val_loss: 0.2568 - val_accuracy:
0.9311 - lr: 0.0010
Epoch 24/25
63/63 [==============================] - ETA: 0s - loss: 0.4229 - accuracy: 0.8601
Epoch 24: val_loss did not improve from 0.14275
63/63 [==============================] - 24s 387ms/step - loss: 0.4229 - accuracy: 0.8601 - val_loss: 1.0645 - val_accuracy:
0.6738 - lr: 0.0010
Epoch 25/25
63/63 [==============================] - ETA: 0s - loss: 0.3869 - accuracy: 0.8751
Epoch 25: val_loss did not improve from 0.14275
63/63 [==============================] - 21s 341ms/step - loss: 0.3869 - accuracy: 0.8751 - val_loss: 0.3520 - val_accuracy:
0.8576 - lr: 0.0010
```

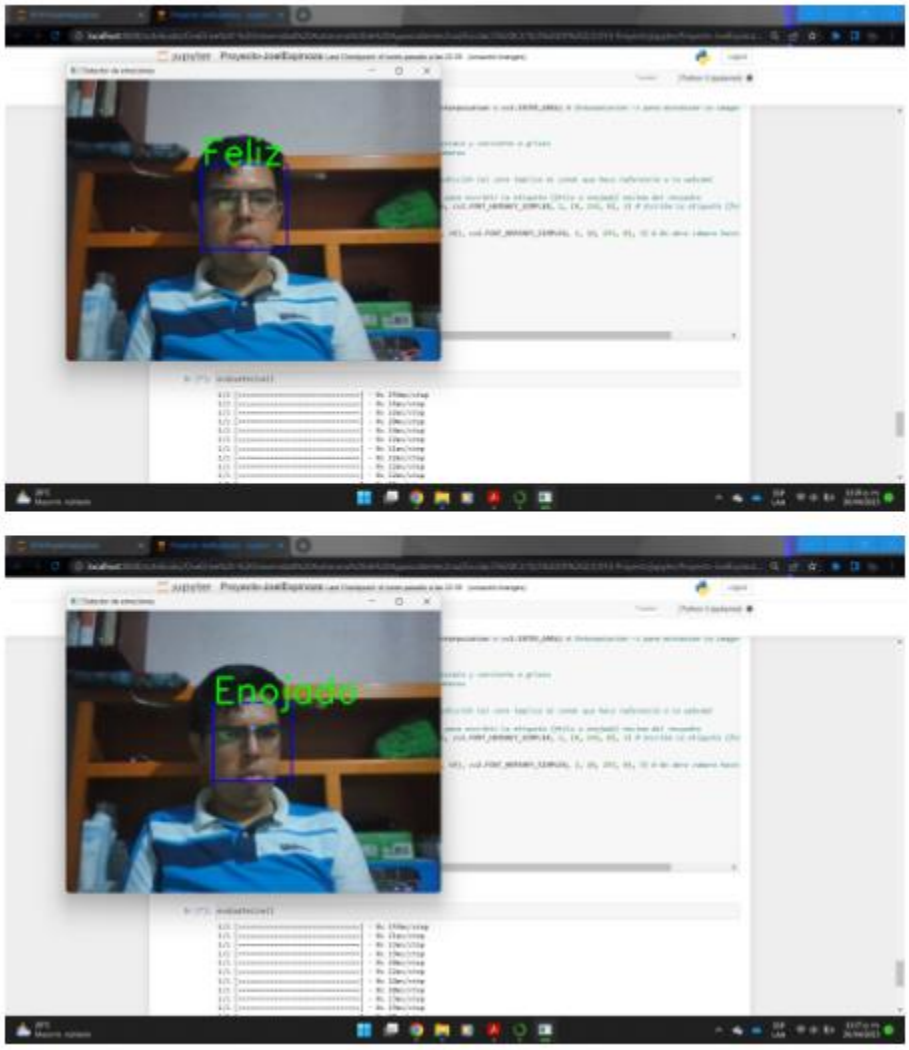
Finalmente se realizó la ejecución de lectura de cámara a tiempo real en la siguiente función que evaluaría el sentimiento con el que se entrenó previamente:

```python
from keras.models import load_model
from time import sleep
from tensorflow.keras.utils import img_to_array
from keras.preprocessing import image
import cv2
import numpy as np

def evaluateLive():
    #
    face_classifier = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml') # Carga Haarcascades
    classifier = load_model('Modelo_3.h5') # Carga modelo
    #class_labels = ['Feliz', 'Enojado', 'Sorpresa', 'Neutral', 'Triste']
    class_labels = ['Enojado', 'Feliz', 'Neutral','Sorprendido', 'Triste']

    # Para cargarle imágenes
    '''
    '''

    # Detección en tiempo real con webcam
    cap = cv2.VideoCapture(0) # Webcam
    while True:
        ret, frame = cap.read() # Lee webcam
        # Preprocesa la imagen
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # escala de grises
        faces = face_classifier.detectMultiScale(gray, 1.3, 5) # Detección de caras (ubicación de la cara)

        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2) # Obtiene sólo el área de la cara (recorta la cara)

            roi_gray = gray[y:y + h, x:x + w] # Vuelve a generar la escala de grises
            roi_gray = cv2.resize(roi_gray, (48, 48), interpolation = cv2.INTER_AREA) # Interpolación -> para minimizar la image

            # Si detecta rostro genera un recuadro
            if np.sum([roi_gray]) != 0:
                roi = roi_gray.astype('float')/255.0 # escala y convierte a grises
                roi = img_to_array(roi) # Convierte a números
                roi = np.expand_dims(roi, axis = 0)

                preds = classifier.predict(roi)[0] # Predicción (el cero implica el canal que hace referencia a la webcam)
                label = class_labels[preds.argmax()]
                label_position = (x, y) # Posición x, y para escribir la etiqueta (feliz o enojado) encima del recuadro
                cv2.putText(frame, label, label_position, cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 3) # Escribe la etiqueta (fe

            else:
                cv2.putText(frame, 'No Face Found', (20, 60), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 3) # No abre cámara hast

        cv2.imshow('Detector de emociones', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

La ejecución se muestra a continuación:

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
```

Se observa el resultado de ejecutar el anterior código en las siguientes capturas:



Conclusión: Es interesante e importante poder implementar las bases de estos temas para entenderlos en un futuro, pues, posteriormente no basta con sólo importar librerías que realicen el trabajo pesado, ya que, implementar manualmente estos algoritmos nos enseña a qué hay detrás del algoritmo, cómo funciona y poder comprender realmente qué está ocurriendo como la base de una red neuronal convolucional y sus alpicaciones en detección de sentimientos a tiempo real. Es muy útil la implementación de estos algoritmos en estas tareas para las futuras tareas de la materia y aplicaciones de Machine Learning en la vida personal.

Referencias:

- Anónimo (s.f.) "*Red neuronal artificial*". Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Red_neuronal_artificial.
- Data Scientest (2021) "*Perceptrón. ¿Qué es y para qué sirve?*". Obtenido de Data Scientest: https://datascientest.com/es/perceptron-que-es-y-para-que-sirve.
- Luna, F. (2023) "*El Modelo de McCulloch – Pitts*". Apuntes de ICI 10°.