**Detección de Emociones en Tiempo Real**



**Dairon Tadeo García Medina   
Eduardo Isaí López García  
Jesús Yocsan Luévano Flores  
Martín Isaí Núñez Villeda  
Alan Fernando Martinez Moreno  
Pablo David Pérez López**

17/11/2024

3er Semestre de Ingeniería en Computación Inteligente

Inteligencia Artificial

Francisco Javier Luna Rosas

Universidad Autónoma de Aguascalientes

**INTRODUCCIÓN**

El reconocimiento facial no es algo nuevo en el mundo actual, de hecho es tan común que varios dispositivos electrónicos como celulares o cerraduras de seguridad lo utilizan desde hace muchos años como parte principal de su funcionamiento, reconociendo patrones en las imágenes con las que se entrena, que es precisamente lo que se intenta realizar en esta práctica de investigación y aplicación. Pero hay una función que en este caso es la función principal de la práctica, que es no solo reconocer rostros, sino también identificar sus emociones solo con las expresiones faciales que realice el usuario.

SI bien en la época actual en la que vivimos, con el auge desmedido de la I.A. y su constante y exponencial evolución, ya existen ciertas tecnologías que facilitan aún más trabajos como este, como lo son las “Redes Neuronales Convolucionales”; pero no se usará nada de eso. A base de redes neuronales “simples” o más primitivas, se logra el mismo objetivo, entrenar al programa con cierta cantidad de imágenes para detectar patrones en distintos tipos de caras y así lograr tener una suficiente cantidad de datos de cada emoción para que el programa sea capaz de identificarlos por cuenta propia y en tiempo real.

Si bien la utilidad de este tipo de tecnologías puede no ser tan clara o tan técnica como otras que hemos visto, puede ayudar y servir muy bien para, por ejemplo, reducir el estrés después de un mal día de trabajo; la I.A. al ser capaz de identificar estas emociones, si llegas triste o enojado, podría tener la capacidad de poner en automático música relajante, podría prender la televisión o preparar un café, o si llegas feliz por ejemplo, puede ayudar a mantener el ánimo más tiempo al poner música movida o inclusive con I.A.’s más avanzadas, planear actividades para ese día. Así que efectivamente, puede no ser tan técnicamente práctico, pero puede ser de gran ayuda para reducir los pesos de un día común, o mantener un estado de ánimo positivo, lo que a la larga ayudaría a la persona a estar más tranquila o ser más feliz con su vida cotidiana.

**ANÁLISIS**

**Algoritmos de Detección de Emociones**

* **Desempeño General**: Al programa se le pasan una serie de datos, en este caso clasificados como enojado, triste, feliz y neutral, siendo estos datos imágenes expresando las emociones. Habiéndole pasado ya las imágenes al extractor de rostros, se pasa al creador de modelo, donde se entrena para que busque patrones entre las caras y así aprenda a identificar nuevas, y finalmente se carga el modelo. Como se puede verificar, son 3 programas o 3 códigos como parte del proyecto.
* **Metodología:** Los datos utilizados para esta práctica son imágenes, y al requerir un volumen grande de imágenes lo que se hizo fue tomar videos de 10 segundos con un aproximado de 30 fps, esto se realizó 6 veces por cada emoción, dando aproximadamente 7,200 imágenes que se utilizaron para el entrenamiento del modelo.
* **Complejidad**: La complejidad del modelo radica en al dificultad de replicar el comportamiento de una red convolucional utilizando redes neuronales normales, por lo que en este caso, para poder aplicarlo más eficientemente, se dividió en 3 códigos.
* **Aplicaciones y Limitaciones**: Aplicaciones no tiene demasiadas, más que las ya mencionadas y puede llegar a ser útil en sistemas de seguridad más avanzados (donde tengas que tener cierta expresión facial específica por ejemplo). Sin embargo, está bastante limitado si es que se quiere hacer un algoritmo de reconocimiento facial general, ya que se requieren grandes volúmenes de datos por cada una de las emociones a identificar (que son muchas las que existen), por lo que para su utilidad, no vale la pena hacer una algoritmo general para este programa.

**IMPLEMENTACIÓN**

Extractor de Rostros

*import* cv2

*import* numpy *as* np

*import* tkinter *as* tk

*from* tkinter *import* messagebox

*import* os

*from* datetime *import* datetime

root *=* tk.Tk()

root.title("Emotion Detection Recopiler")

selected\_option *=* tk.StringVar()

source\_option *=* tk.StringVar(value*=*"Camera")

*# Define variables to store the path and type globally*

ruta\_base *=* "Extraccion\_Emociones/"

ruta *=* ""

tipo *=* ""

*def* *on\_go\_button*():

*global* ruta, tipo

option *=* selected\_option.get()

*if* option:

tipo *=* process\_selected\_option(option)

create\_directory() *# Create directory based on the current date and time*

messagebox.showinfo("Selected Option", *f*"You selected: {option}")

root.destroy() *# Close the main window after selection*

*else*:

messagebox.showwarning("No Selection", "Please select an option")

*def* *process\_selected\_option*(option):

*global* tipo

*if* option *==* "Happy":

print("Processing Happy images...")

tipo *=* "Feliz"

*elif* option *==* "Sad":

print("Processing Sad images...")

tipo *=* "Triste"

*elif* option *==* "Angry":

print("Processing Angry images...")

tipo *=* "Enojado"

*return* tipo

*def* *create\_directory*():

*global* ruta

*# Get the current date and time*

now *=* datetime.now()

*# Format the date and time as a string*

directory\_name *=* now.strftime("%Y-%m-%d\_%H-%M-%S")

*# Create the full path*

ruta *=* os.path.join(ruta\_base, tipo, directory\_name)

*# Create the directory*

os.makedirs(ruta, exist\_ok*=*True)

*# Create radio buttons for emotion selection*

radio1 *=* tk.Radiobutton(root, text*=*"Happy", variable*=*selected\_option, value*=*"Happy", bg*=*'black', fg*=*'white')

radio2 *=* tk.Radiobutton(root, text*=*"Sad", variable*=*selected\_option, value*=*"Sad", bg*=*'black', fg*=*'white')

radio3 *=* tk.Radiobutton(root, text*=*"Angry", variable*=*selected\_option, value*=*"Angry", bg*=*'black', fg*=*'white')

*# Pack the radio buttons*

radio1.pack(anchor*=*tk.W)

radio2.pack(anchor*=*tk.W)

radio3.pack(anchor*=*tk.W)

*# Create radio buttons for source selection*

camera\_radio *=* tk.Radiobutton(root, text*=*"Camera", variable*=*source\_option, value*=*"Camera", bg*=*'black', fg*=*'white')

video\_radio *=* tk.Radiobutton(root, text*=*"Video", variable*=*source\_option, value*=*"Video", bg*=*'black', fg*=*'white')

*# Pack the source radio buttons*

camera\_radio.pack(anchor*=*tk.W)

video\_radio.pack(anchor*=*tk.W)

*# Create and pack the "Go" button*

go\_button *=* tk.Button(root, text*=*"Go", command*=*on\_go\_button, bg*=*'orange', fg*=*'white')

go\_button.pack()

*# Run the application*

root.mainloop()

*# Decide the video source based on user's choice*

*if* source\_option.get() *==* "Camera":

cap *=* cv2.VideoCapture(0)

*else*:

cap *=* cv2.VideoCapture("truste\_2.mp4")

face\_cascade *=* cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

i *=* 0

*while* True:

ret, frame *=* cap.read()

*if* *not* ret:

*break*

gray *=* cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

framecopy *=* frame.copy()

faces *=* face\_cascade.detectMultiScale(gray, scaleFactor*=*1.2, minNeighbors*=*5)

*for* (x, y, w, h) *in* faces:

cv2.rectangle(frame, (x, y), (x*+*w, y*+*h), (0, 255, 255), 2)

rostro *=* framecopy[y:y*+*h, x:x*+*w]

rostro *=* cv2.resize(rostro, (150, 150), interpolation*=*cv2.INTER\_CUBIC)

*# Save the image with a correct path and file name*

cv2.imwrite(*f*"{ruta}/{tipo}\_{i}.jpg", rostro)

i *+=* 1

cv2.imshow("Video", frame)

k *=* cv2.waitKey(1)

*if* k *==* ord("q") *or* i *>=* 300:

*break*

cap.release()

cv2.destroyAllWindows()

Creador de Modelo

*import* os

*import* numpy *as* np

*from* PIL *import* Image

*# Function to load images from a folder and subfolders*

*def* *load\_images\_from\_folder*(folder, label\_map):

images *=* []

labels *=* []

*for* label\_name, label *in* label\_map.items():

label\_folder *=* os.path.join(folder, label\_name)

*for* subfolder\_name *in* os.listdir(label\_folder):

subfolder\_path *=* os.path.join(label\_folder, subfolder\_name)

*if* os.path.isdir(subfolder\_path):

*for* idx, filename *in* enumerate(os.listdir(subfolder\_path)):

img\_path *=* os.path.join(subfolder\_path, filename)

img *=* Image.open(img\_path).convert('L')

img *=* img.resize((48, 48))

img\_array *=* np.array(img).astype(np.float32).flatten()

images.append(img\_array)

labels.append(label)

*return* np.array(images), np.array(labels)

*# Neural Network Class Definition*

*class* SimpleNN:

*def* \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

*self*.W1 *=* np.random.randn(input\_size, hidden\_size) *\** 0.01

*self*.b1 *=* np.zeros((1, hidden\_size))

*self*.W2 *=* np.random.randn(hidden\_size, output\_size) *\** 0.01

*self*.b2 *=* np.zeros((1, output\_size))

*def* *sigmoid*(self, z):

*return* 1 */* (1 *+* np.exp(*-*z))

*def* *sigmoid\_derivative*(self, z):

s *=* *self*.sigmoid(z)

*return* s *\** (1 *-* s)

*def* *forward*(self, X):

*self*.Z1 *=* np.dot(X, *self*.W1) *+* *self*.b1

*self*.A1 *=* *self*.sigmoid(*self*.Z1)

*self*.Z2 *=* np.dot(*self*.A1, *self*.W2) *+* *self*.b2

*self*.A2 *=* *self*.sigmoid(*self*.Z2)

*return* *self*.A2

*def* *compute\_cost*(self, Y, A2):

m *=* Y.shape[0]

cost *=* *-*np.sum(Y *\** np.log(A2) *+* (1 *-* Y) *\** np.log(1 *-* A2)) */* m

*return* cost

*def* *backward*(self, X, Y):

m *=* X.shape[0]

dZ2 *=* *self*.A2 *-* Y

dW2 *=* np.dot(*self*.A1.T, dZ2) */* m

db2 *=* np.sum(dZ2, axis*=*0, keepdims*=*True) */* m

dA1 *=* np.dot(dZ2, *self*.W2.T)

dZ1 *=* dA1 *\** *self*.sigmoid\_derivative(*self*.Z1)

dW1 *=* np.dot(X.T, dZ1) */* m

db1 *=* np.sum(dZ1, axis*=*0, keepdims*=*True) */* m

*self*.W1 *-=* *self*.learning\_rate *\** dW1

*self*.b1 *-=* *self*.learning\_rate *\** db1

*self*.W2 *-=* *self*.learning\_rate *\** dW2

*self*.b2 *-=* *self*.learning\_rate *\** db2

*def* *train*(self, X, Y, epochs, learning\_rate):

*self*.learning\_rate *=* learning\_rate

*for* i *in* range(epochs):

A2 *=* *self*.forward(X)

cost *=* *self*.compute\_cost(Y, A2)

*self*.backward(X, Y)

*if* i *%* 100 *==* 0:

print(*f*"Iteration {i}, Cost: {cost}")

*def* *save\_model*(self, file\_path):

np.savez(file\_path, W1*=self*.W1, b1*=self*.b1, W2*=self*.W2, b2*=self*.b2)

*@*staticmethod

*def* *load\_model*(file\_path):

data *=* np.load(file\_path)

model *=* SimpleNN(data['W1'].shape[0], data['W1'].shape[1], data['W2'].shape[1])

model.W1 *=* data['W1']

model.b1 *=* data['b1']

model.W2 *=* data['W2']

model.b2 *=* data['b2']

*return* model

*# Main Execution \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# Load and preprocess dataset*

dataset\_folder *=* 'Extraccion\_Emociones' *# Change this to your dataset folder path*

label\_map *=* {'Feliz': 0, 'Triste': 1, 'Enojado': 2}

X, y *=* load\_images\_from\_folder(dataset\_folder, label\_map)

X *=* X */* 255.0

y *=* np.eye(len(label\_map))[y]

*# Split the data into training and testing sets*

*from* sklearn.model\_selection *import* train\_test\_split

X\_train, X\_test, y\_train, y\_test *=* train\_test\_split(X, y, test\_size*=*0.2, random\_state*=*42)

*# Initialize and train the neural network*

input\_size *=* 48 *\** 48

hidden\_size *=* 64

output\_size *=* 3

nn *=* SimpleNN(input\_size, hidden\_size, output\_size)

learning\_rate *=* 0.01

epochs *=* 10000

nn.train(X\_train, y\_train, epochs, learning\_rate)

*# Predict function*

*def* *predict*(X, model):

A2 *=* model.forward(X)

*return* np.argmax(A2, axis*=*1)

*# Evaluate the model*

train\_predictions *=* predict(X\_train, nn)

test\_predictions *=* predict(X\_test, nn)

train\_accuracy *=* np.mean(np.argmax(y\_train, axis*=*1) *==* train\_predictions)

test\_accuracy *=* np.mean(np.argmax(y\_test, axis*=*1) *==* test\_predictions)

print(*f*"Train Accuracy: {train\_accuracy *\** 100}%")

print(*f*"Test Accuracy: {test\_accuracy *\** 100}%")

*# Save the trained model*

nn.save\_model('emotion\_detection\_model.npz')

Modelo

*import* numpy *as* np

*import* cv2

*# Define the SimpleNN class*

*class* SimpleNN:

*def* \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

*self*.W1 *=* np.random.randn(input\_size, hidden\_size) *\** 0.01

*self*.b1 *=* np.zeros((1, hidden\_size))

*self*.W2 *=* np.random.randn(hidden\_size, output\_size) *\** 0.01

*self*.b2 *=* np.zeros((1, output\_size))

*def* *sigmoid*(self, z):

*return* 1 */* (1 *+* np.exp(*-*z))

*def* *forward*(self, X):

*self*.Z1 *=* np.dot(X, *self*.W1) *+* *self*.b1

*self*.A1 *=* *self*.sigmoid(*self*.Z1)

*self*.Z2 *=* np.dot(*self*.A1, *self*.W2) *+* *self*.b2

*self*.A2 *=* *self*.sigmoid(*self*.Z2)

*return* *self*.A2

*# Load model function*

*def* *load\_model*(file\_path, input\_size, hidden\_size, output\_size):

data *=* np.load(file\_path)

model *=* SimpleNN(input\_size, hidden\_size, output\_size)

model.W1 *=* data['W1']

model.b1 *=* data['b1']

model.W2 *=* data['W2']

model.b2 *=* data['b2']

*return* model

*# Parameters (adjust as needed based on your trained model)*

input\_size *=* 48 *\** 48 *# Image size (48x48 pixels flattened)*

hidden\_size *=* 64 *# Number of neurons in the hidden layer*

output\_size *=* 3 *# Number of output classes (emotions)*

*# Load the trained model*

nn *=* load\_model('emotion\_detection\_model.npz', input\_size, hidden\_size, output\_size)

*# Main testing with real-time video*

cap *=* cv2.VideoCapture(0)

face\_cascade *=* cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

*while* True:

ret, frame *=* cap.read()

gray *=* cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces *=* face\_cascade.detectMultiScale(gray, scaleFactor*=*1.2, minNeighbors*=*5)

*for* (x, y, w, h) *in* faces:

cv2.rectangle(frame, (x, y), (x*+*w, y*+*h), (0, 255, 0), 2)

face *=* gray[y:y*+*h, x:x*+*w]

face *=* cv2.resize(face, (48, 48)).flatten().astype(np.float32) */* 255.0

face *=* face.reshape(1, *-*1)

prediction *=* nn.forward(face)

emotion *=* np.argmax(prediction)

*if* emotion *==* 0:

label *=* 'Happy'

*elif* emotion *==* 1:

label *=* 'Sad'

*elif* emotion *==* 2:

label *=* 'Angry'

cv2.putText(frame, label, (x, y*-*10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 255, 0), 2)

cv2.imshow("Video", frame)

*if* cv2.waitKey(1) *&* *0x*FF *==* ord('q'):

*break*

cap.release()

cv2.destroyAllWindows()

## Explicación del Extractor de Rostros:

1. Inicialización: Se inicializa una interfaz gráfica para el modelo, y se inicializan las variables globales que se utilizarán para el manejo de rutas y directorios.
2. Operación Principal: Se captura el video y se procesa cada fotograma convirtiendo el cuadro a escala de grises, luego se detecta el rostro, se identifican sus características, y se guarda en una carpeta.
3. Procesamientos Adicionales: Si hay más de un rostro, se procesan por separado; y las carpetas se organizan por fecha, hora y emoción.
4. Resultado: El bucle termina hasta que se hayan capturado 300 imágenes, y se espera que hayan identificado correctamente los rostros en las imágenes..

## Explicación del Creador de Modelo:

1. Inicialización: En este código se cargan las imágenes sacadas por el Extractor de Rostros, se procesan en una red neuronal con 64 neuronas en su capa oculta, y se utiliza la librería de “train\_test\_split” para entrenar al modelo.
2. Operación Principal: Se utiliza primero forward propagation para calcular la salida de la red neuronal, se calcula el error entre la salida y una función de pérdida, y finalmente se utiliza backpropagation para la actualización de pesos.
3. Procesamientos Adicionales: Para el entrenamiento se utilizaron 10,000 épocas, y una vez entrenado se utiliza la función “predict” para luego calcular la precisión comparando las salidas predichas con salidas previamente etiquetadas.
4. Resultado: Lo que se espera en este código es que nos de los porcentajes de precisión, esperando que sean altos.

## Explicación del Modelo:

1. Inicialización: Al ser la aplicación del Creador del Modelo, simplemente se define la red neuronal, inicializando valores; luego se prepara la captura de video y al último se definen los parámetros que se utilizarán en la red neuronal.
2. Operación Principal: La operación principal es la aplicación práctica de los 2 anteriores códigos, así que captura las imágenes, detecta los rostros, preprocesa las nuevas imágenes, y finalmente predice las emociones utilizando los parámetros ya establecidos.
3. Procesamiento Adicional: Todo este proceso se hace en tiempo real, pudiendo ver el video mientras predice las emociones al mismo tiempo.
4. Resultado: Se espera que prediga de forma correcta las emociones que uno está expresando en el momento.

**EVALUACIÓN DEL ALGORITMO**

**Descripción:**El algoritmo se basa en redes neuronales y librerías de procesamiento de imágenes cuyo objetivo final es reconocer las emociones humanas en tiempo real por medio de las expresiones faciales que realiza el usuario.

**Pros:**

* Simplicidad: Si bien es más tardado que utilizar redes convolucionales, es sencillo de entender al utilizar conceptos menos complejos que una red convolucional.
* Capacidad de procesamiento: Si bien en este caso se entrenó para 4 emociones, no es demasiado complicado agregar las emociones que se requieran, solo hay que modificar un poco el código.

**Contras:**

* Volúmenes de datos: A pesar de su “sencillez”, los volúmenes de datos que se requieren son demasiado altos, simplemente para 4 emociones se requirieron más de 7,000 imágenes.
* Generalización del algoritmo: Relacionado con la cantidad de datos, incluso con esas 7,000 imágenes, no es 100% seguro que funcione para cada una de las personas que intenten utilizarlo, ya que utilizamos solamente 6 modelos de rostros, así que se requieren una cantidad inmensa de datos para generalizar el algoritmo.

**Complejidad:**

* Tiempo: Es un código algo complejo de aplicar, ya que por ejemplo, no se utilizó un solo código, sino que se tuvo que separar en 3 códigos para que fuera más sencillo de aplicar y entender..

Aplicaciones Recomendadas:  
Principalmente como práctica de varios conceptos de I.A. y aprendizaje, ya que sus aplicaciones prácticas son complicadas de concretar.

**CONCLUSIONES**

El proyecto nos ayudó a comprender de mucho mejor forma el funcionamiento de las redes neuronales y manipulación de archivos e imagenes, cosa que representó un desafío muy grande en cuestión de creación de contenido, ya qué el manejar datasets de ese tamaño es demasiado grande.

Una vez teniendo ya los datasets, fue un poco más sencillo el realizar los códigos qué sé utilizaron, aunque si se tuvo que modificar de forma en que se pudiesen llegar a los objetivos, lo cual no fue algo demasiado complicado teniendo en cuenta los conocimientos previos.

En conclusión, está práctica nos ayudó mucho a reafirmar y practicar todo lo que ya hemos visto en el curso del semestre, y representó un reto no solo en lo practico sino en la organización en general del proyecto, por lo que fue un gran proyecto en cuanto al reto qué nos generó.

**REFERENCIAS**

Kuksov, I. (2020, 2 abril). Sistemas de reconocimiento de emociones: ¿Queremos que la IA nos domine? *Kaspersky Daily*. <https://latam.kaspersky.com/blog/emotional-ai/16645/>