Instituto Tecnológico campus Culiacán

Inteligencia Artificial

Carrera:

Ingeniería en Sistemas Computacionales

Actividad:

Creación, entrenamiento y funcionamiento de un visor IA

Alumno:

21170293 Fernando Chiquete Velazquez

21170387 Omar Manjarrez Rodelo

Maestr@:

Zuriel Dathan Mora Felix

29/05/2025

Detalle

Dentro de esta tarea se creo un visor de inteligencia artificial el cual detecte emociones. La tarea consistía en realizar un visor que detecte por medio de nuestra cámara la emoción que tenemos en ese momento, además de visualizar que tan seguro esta de este análisis.

Algunos inconvenientes que notamos al realizar este proyecto fueron que las librerías que intentábamos utilizar de TensorFlow no eran compatibles con las versiones de Python que nosotros contábamos por lo que decidimos utilizar un ambiente virtual para la instalación y posteriormente el entrenamiento del modelo.

Además de que tuvimos que realizar de nuevo el preprocesamiento de las imágenes ya que como lo habíamos hecho la tarea anterior de agregar borrosidad a nuestras imágenes no nos daban resultados del todo favorables es por ello que tras una pequeña investigación nos dimos cuenta que agregar escala de grises a nuestro data set era conveniente a la borrosidad.

Código

Para empezar a entrenar el modelo lo que aplicamos fue a aplicarle los filtros a las imágenes del dataset, primero tomamos en cuenta que fueran de un tamaño 100x100 para después aplicarle una escala de grises que nos permitiera hacer un contraste mayor.

```
class EmotionModelTrainer:
    def __init__(self):
        self.img_size = 100
        self.batch_size = 32
        self.epochs = 15
        self.train_dir = 'DATASET/train'
        self.test_dir = 'DATASET/test'
        self.model = None
        self.history = None

    def setup_data_generators(self):
        train_datagen = ImageDataGenerator(rescale=1./255)
        test_datagen = ImageDataGenerator(rescale=1./255)

        self.train_generator = train_datagen.flow_from_directory(
            self.train_dir,
            target_size=(self.img_size, self.img_size),
            batch_size=self.batch_size,
            color_mode='grayscale',
            class_mode='categorical'
        )

        self.test_generator = test_datagen.flow_from_directory(
            self.test_dir,
            target_size=(self.img_size, self.img_size),
            batch_size=self.batch_size,
            color_mode='grayscale',
            class_mode='categorical',
            shuffle=False
```

Dentro del siguiente fragmento de código creamos la CNN, la cual constara de 3 capas las cuales aplicaran filtros a la imagen, en cada capa el número de filtros crecerá, haciendo que cada vez haya características mas complejas y el entrenamiento de nuestra red sea más preciso.

Además de utilizar el método Adam para que la red ajuste sus pesos y sea mas adecuado sus valores, así como también integrar una etiqueta de precisión para denotar que tan bueno fue el modelo.

```
def build_model(self):
    self.model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(self.img_size, self.img_size, 1)),
        MaxPooling2D(2, 2),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),

        Dense(self.train_generator.num_classes, activation='softmax')
    ])

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

Por último de lo que corresponde al paquete de modelo tenemos la ejecución en la cual ejecutamos el modelo y ponemos en marcha el entrenamiento de la red convolucional así como también generamos un reporte de cuales fueron los índices de aceptación de cada emoción.

```
    def evaluate_model(self):
        Y_pred = self.model.predict(
            self.test_generator,
            steps=self.test_generator.samples // self.test_generator.batch_size + 1
        )
        y_pred = np.argmax(Y_pred, axis=1)
        y_true = self.test_generator.classes

        target_names = list(self.test_generator.class_indices.keys())

        cm = confusion_matrix(y_true, y_pred)
        print("Reporte de clasificación:")
        print(classification_report(y_true, y_pred, target_names=target_names))
        self.plot_confusion_matrix(cm, target_names)

    def run(self):
        self.setup_data_generators()
        self.build_model()
        self.train_model()
        self.evaluate_model()

if __name__ == "__main__":
    trainer = EmotionModelTrainer()
    trainer.run()
```

```
Reporte de Clasificación:
              precision    recall  f1-score   support

       happy       0.99      1.00      0.99      5161
         sad       1.00      0.99      0.99      4938

    accuracy                           0.99     10099
   macro avg       0.99      0.99      0.99     10099
weighted avg       0.99      0.99      0.99     10099

{'happy': 0, 'sad': 1}
```

Ahora viene la ejecución del main, el cual nos crea una ventana que la cual nos muestra la cámara de nuestra computadora, detecta nuestro rostro y por medio de etiquetas nos dice si la emoción representada por nosotros es “Happy” o “Sad” además de decirnos que tan preciso es esta predicción. Pero para eso tenemos que hacer:

```python
class EmotionDetector:
    def __init__(self, model_path="modelo_emociones.h5"):
        self.model = load_model(model_path)
        self.labels = ['happy', 'sad']
        self.face_cascade = cv2.CascadeClassifier(
            cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
        )
        self.img_size = 100
        self.cap = cv2.VideoCapture(0)
        self.font = cv2.FONT_HERSHEY_SIMPLEX
```

Lo primero que hacemos es la carga del método, definimos las etiquetas que queremos que nos muestre, después realizamos que nuestra dimensión de video sea de 100x100, después de esto iniciamos la cámara de nuestro equipo y cargamos la fuente que deseamos que aparezca.

Lo consiguiente a hacer es el convetir lo que nuestra cámara esta viendo en un arreglo para que sea mas sencillo el estudio de esta eso es lo que hace el método preprocess_fase, en predict_emotion lo que realiza es ir a buscar dentro de nuestro modelo la probabilidad a la que tenga más cercanía y el resultado será mostrada en la etiqueta que aparecerá en nuestra pantalla.

Por último el método de draw_results solo nos dibujara el recuadro azul que tenemos en nuestra cara delimitando en donde esta nuestro rostro, y agregara las etiquetas de emoción y seguridad de esta elección.

```python
def preprocess_face(self, face_image):
    face_image = cv2.resize(face_image, (self.img_size, self.img_size))
    face_image = face_image.astype("float") / 255.0
    face_image = img_to_array(face_image)
    return np.expand_dims(face_image, axis=0)

def predict_emotion(self, face_roi):
    prediction = self.model.predict(face_roi)[0]
    label = self.labels[np.argmax(prediction)]
    confidence = np.max(prediction)
    return label, confidence

def draw_result(self, frame, x, y, w, h, label, confidence):
    cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cv2.putText(frame, f"{label} ({confidence:.2f})",
                (x, y-10), self.font, 0.8, (255, 255, 255), 2)

def run(self):
```