Instituto Tecnológico Nacional de México

Instituto Tecnológico de Culiacán

Ingeniería en sistemas computacionales

Inteligencia Artificial

Docente: Zuriel Dathan Mora Felix

Reconocimiento de emociones

Unidad IV

Integrantes:

- Amarillas Aviles Brayan Alexis
- Cuen Armenta Alma Victoria

índice

1 – Selección de modelo

Se seleccionó la red neuronal convolucional (CNN) como modelo principal debido a su efectividad en el procesamiento de imágenes y reconocimiento de patrones faciales. Como equipo, consideramos que esta red es ideal para el proyecto, ya que permite extraer automáticamente las características más relevantes de las expresiones y clasificar las emociones con alta precisión. Gracias a la CNN, nuestro sistema puede analizar rostros de manera eficiente y proporcionar resultados confiables en el reconocimiento emocional.

2- Adquisicion de datos y anotacion

Los datos utilizados en este proyecto fueron recopilados del conjunto de datos **FER-2013**, disponible en la plataforma **Kaggle**. Este dataset contiene imágenes de rostros en escala de grises, etiquetadas con distintas emociones, y fue utilizado como base para el entrenamiento y evaluación del modelo.

https://www.kaggle.com/datasets/msambare/fer2013

2.1 Procesamiento de imágenes

Se aplicaron transformaciones a las imágenes utilizando torchvision.transforms para normalizar y aumentar la variabilidad del dataset. Las imágenes se convirtieron a escala de grises, se redimensionaron a 48x48 píxeles, se aplicaron rotaciones aleatorias y ajustes de brillo, y finalmente se convirtieron a tensores y se normalizaron.

```python
# Transformaciones para normalizar e igualar tamaño
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),  # Si las imágenes son en escala de grises
    transforms.Resize((48, 48)),                  # Tamaño común en FER-2013
    transforms.RandomRotation(20),                # Rotación aleatoria para aumentar la variabilidad
    transforms.ColorJitter(brightness=(0.8, 1.2)), # Ajuste de brillo
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

✓ 0.0s Python

2.2 Etiquetado de imágenes

No fue necesario realizar un etiquetado manual de las imágenes, ya que el conjunto de datos utilizado está estructurado en carpetas, donde cada subcarpeta representa una clase. Esta organización permite que el framework PyTorch, a través de la función ImageFolder, detecte automáticamente las etiquetas correspondientes según el nombre de cada subdirectorio.

3- Argumentación de datos

El conjunto de datos se carga utilizando ImageFolder, lo cual simplifica la organización ya que las etiquetas se asignan automáticamente según el nombre de las carpetas. Posteriormente, se emplea DataLoader para dividir los datos en lotes de tamaño 64, aplicando aleatorización al conjunto de entrenamiento.

```python
train_dataset = datasets.ImageFolder(os.path.join(data_dir, 'train'), transform=transform)
test_dataset = datasets.ImageFolder(os.path.join(data_dir, 'test'), transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

✓ 0.1s Python

4-Entrenar el modelo

Se definió la arquitectura del modelo **EmotionCNN**, una red neuronal convolucional compuesta por tres bloques de capas Conv2D, cada uno seguido por funciones de activación ReLU y capas de MaxPool2d, lo que permite extraer y reducir las características más relevantes de las imágenes. Para la etapa de clasificación, se utilizaron capas Flatten, Linear y Dropout, con el objetivo de transformar las características extraídas en una predicción correspondiente a una de las clases emocionales definidas.

Como parte del entrenamiento, se implementó la función de pérdida CrossEntropyLoss, ideal para tareas de clasificación multiclase, y se empleó el optimizador Adam, que ajusta los pesos del modelo de forma eficiente para minimizar la pérdida durante el aprendizaje.

```python
class EmotionCNN(nn.Module):
    def __init__(self, num_classes=5):
        super(EmotionCNN, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 6 * 6, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.conv(x)
        x = self.fc(x)
        return x
```

✓ 0.0s Python

Primero detecta si hay una GPU disponible para usarla; si no, entrena con la CPU. Define la función de pérdida (CrossEntropyLoss) y el optimizador (Adam). Luego, durante 15 épocas, alimenta imágenes al modelo, calcula la pérdida, realiza retropropagación para ajustar los pesos, y muestra la pérdida promedio por época para evaluar el progreso del entrenamiento.

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = EmotionCNN(num_classes=len(train_dataset.classes)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Entrenar por 15 épocas
for epoch in range(15):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Época {epoch+1}, pérdida: {running_loss/len(train_loader):.4f}")
```

✓ 5m 20.7s Python

```
Época 1, pérdida: 1.3171
Época 2, pérdida: 1.1056
Época 3, pérdida: 1.0018
Época 4, pérdida: 0.9339
Época 5, pérdida: 0.8835
Época 6, pérdida: 0.8424
Época 7, pérdida: 0.8119
Época 8, pérdida: 0.7827
Época 9, pérdida: 0.7567
Época 10, pérdida: 0.7434
Época 11, pérdida: 0.7168
Época 12, pérdida: 0.7030
Época 13, pérdida: 0.6832
Época 14, pérdida: 0.6650
Época 15, pérdida: 0.6520
```

5-Evaluacion

Evalúa el modelo con datos de prueba. Desactiva el cálculo de gradientes para ahorrar recursos (torch.no_grad()), pasa las imágenes por el modelo para obtener predicciones, compara esas predicciones con las etiquetas reales y cuenta cuántas fueron correctas. Al final, calcula y muestra la precisión del modelo, que en este caso es del 71.06%.

```
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Precisión en el conjunto de prueba: {100 * correct / total:.2f}%')
```

✓ 2.1s Python

```
Precisión en el conjunto de prueba: 71.06%
```

5.1 Prueba 1

Se carga la imagen prueba.jpg usando OpenCV y se convierte a escala de grises para simplificar el procesamiento visual.

```
# Clases
classes = train_dataset.classes

# Ruta de la imagen
img_path = "prueba.jpg"

# Cargar imagen y convertir a escala de grises
img = cv2.imread(img_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Se utiliza un clasificador Haar Cascade para detectar rostros en la imagen. Si se encuentra alguno, se recorta la región del rostro para enfocarse solo en esa área.

```
# Opcional: detectar rostro y recortar
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

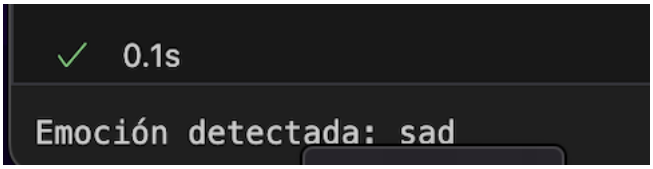
Posteriormente ,se pasa la imagen preprocesada al modelo, que predice la emoción correspondiente .

```python
if len(faces) == 0:
    print("No se detectaron rostros.")
else:
    for (x, y, w, h) in faces:
        roi = gray[y:y+h, x:x+w]
        roi = cv2.resize(roi, (48, 48))
        roi = Image.fromarray(roi)

        # Preprocesar y pasar al modelo
        input_tensor = transform(roi).unsqueeze(0).to(device)

        with torch.no_grad():
            output = model(input_tensor)
            _, predicted = torch.max(output, 1)
            emotion = classes[predicted.item()]
            print(f"Emoción detectada: {emotion}")
```

Resultado final :



5.2 Prueba 2

Se carga la imagen prueba3.jpg con OpenCV utilizando cv2.imread(). Luego, se convierte a escala de grises mediante cv2.cvtColor() para simplificar el procesamiento de la imagen.

```
# Clases
classes = train_dataset.classes

# Ruta de la imagen
img_path = "prueba3.jpg"

# Cargar imagen y convertir a escala de grises
img = cv2.imread(img_path)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Posteriormente se utiliza un clasificador Haar Cascade para detectar rostros en la imagen en escala de grises. Si se encuentran rostros, se recorta la región correspondiente (ROI) para centrarse en la cara.

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray, 1.3, 5)

if len(faces) == 0:
    print("No se detectaron rostros.")
else:
    for (x, y, w, h) in faces:
        roi = gray[y:y+h, x:x+w]
        roi = cv2.resize(roi, (48, 48))
        roi = Image.fromarray(roi)
```

La imagen recortada se transforma para adaptarse al modelo: se convierte a tensor, se normaliza y se añade una dimensión adicional para representar el batch.

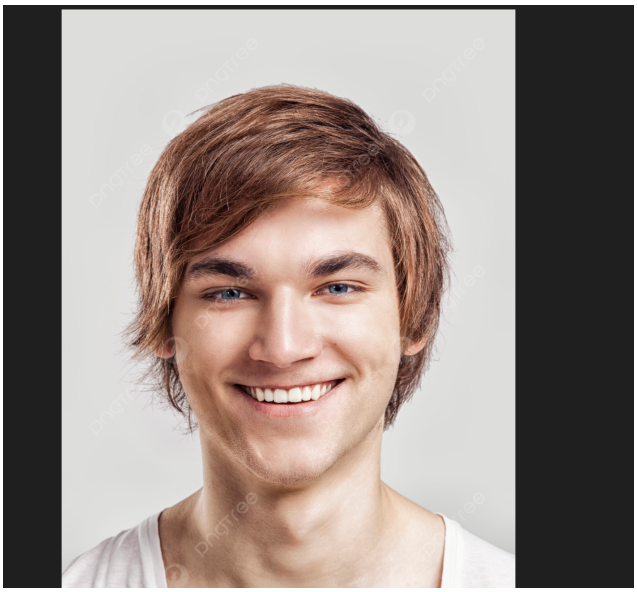
```
        # Preprocesar y pasar al modelo
        input_tensor = transform(roi).unsqueeze(0).to(device)
```

Con torch.no_grad() se desactiva el cálculo de gradientes (no se entrena). La imagen se pasa al modelo para predecir la emoción. Se obtiene la clase con mayor probabilidad y se imprime la emoción detectada.

```
        with torch.no_grad():
            output = model(input_tensor)
            _, predicted = torch.max(output, 1)
            emotion = classes[predicted.item()]
            print(f"Emoción detectada: {emotion}")
```

Resultado final :

```
Emoción detectada: happy
```



Link del video ：

https://drive.google.com/file/d/1lNKRllaND5vvzwaQtsRDnEG8kiQZxj5C/view?usp=share_link