Práctica III: Uso práctico del aumento de datos. Clasificador de estados de ánimo

Introducción

El objetivo de esta práctica es desarrollar y comparar diferentes modelos de Deep Learning para la clasificación de expresiones faciales utilizando el dataset **FER-2013**. El dataset consta de imágenes de rostros clasificadas en 7 clases: *Angry*, *Disgust*, *Fear*, *Happy*, *Neutral*, *Sad*, *Surprise*.

Se han implementado y evaluado tres estrategias:

1. **CNN Propia:** Una red neuronal convolucional simple entrenada desde cero.
2. **Transfer Learning (Extracción de Características):** Uso del modelo **VGGFace** pre-entrenado.
3. **Transfer Learning (Ajuste Fino / Fine Tuning):** Re-entrenamiento de las últimas capas de VGGFace.

Evidencias de realización (código)

Carga y preprocesamiento del dataset

Se descargó el dataset desde Google Drive y se descomprimió. Se utilizaron generadores de datos (*ImageDataGenerator*) para aplicar *Data Augmentation* en el conjunto de entrenamiento y normalización en todos los conjuntos.

Generadores de datos

CNN en grayscale

[ ]

```python
# AUMENTO DE DATOS -> SOLO PARA TRAIN
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    validation_split=0.1
)

# SIN AUMENTO -> PARA VALIDACIÓN Y TEST
validation_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.1
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
[ ]
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=BATCH_SIZE,
    subset="training"
)

val_generator = validation_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=BATCH_SIZE,
    subset="validation"
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    color_mode="grayscale",
    class_mode="categorical",
    batch_size=BATCH_SIZE,
    shuffle=False
)

num_classes = train_generator.num_classes
print("Clases detectadas:", train_generator.class_indices)
```

```
Found 25841 images belonging to 7 classes.
Found 2868 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
Clases detectadas: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4, 'sad': 5, 'surprise': 6}
```

VGGFace (RGB)

```
train_datagen_vgg = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=15,
    zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    validation_split=0.1
)

validation_datagen_vgg = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    validation_split=0.1
)

test_datagen_vgg = ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator_vgg = train_datagen_vgg.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE_VGG,
    color_mode="rgb",
    class_mode="categorical",
    batch_size=BATCH_SIZE_VGG,
    subset="training"
)
```

```
val_generator_vgg = validation_datagen_vgg.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE_VGG,
    color_mode="rgb",
    class_mode="categorical",
    batch_size=BATCH_SIZE_VGG,
    subset="validation"
)

test_generator_vgg = test_datagen_vgg.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE_VGG,
    color_mode="rgb",
    class_mode="categorical",
    batch_size=BATCH_SIZE_VGG,
    shuffle=False
)

num_classes = train_generator_vgg.num_classes
print("Clases detectadas:", train_generator_vgg.class_indices)
```

```
Found 25841 images belonging to 7 classes.
Found 2868 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
Clases detectadas: {'angry': 0, 'disgust': 1, 'fear': 2, 'happy': 3, 'neutral': 4, 'sad': 5, 'surprise': 6}
```

Definición del modelo CNN Simple

Se diseñó una arquitectura sencilla con dos bloques convolucionales seguidos de capas densas.

CNN simple

```
[ ]
cnn_model = Sequential()

# Bloque 1
cnn_model.add(Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(48, 48, 1)))
cnn_model.add(MaxPooling2D(pool_size=(2,2)))

# Bloque 2
cnn_model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
cnn_model.add(MaxPooling2D(pool_size=(2,2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(num_classes, activation='softmax'))

cnn_model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	320
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)	0
conv2d_1 (Conv2D)	(None, 24, 24, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 7)	64,519

```
Total params: 83,335 (325.53 KB)
Trainable params: 83,335 (325.53 KB)
Non-trainable params: 0 (0.00 B)
```

Configuración de VGGFace (extracción de características)

Se importó el modelo base VGGFace, congelando sus pesos y añadiendo un clasificador personalizado en la parte superior.

VGGFace - Extracción de características

```
vggface_base = VGGFace(
    model="vgg16",
    include_top=False,
    input_shape=(224,224,3)
)

vggface_base.trainable = False

vgg_feature_model = Sequential()

vgg_feature_model.add(vggface_base)
vgg_feature_model.add(GlobalAveragePooling2D())
vgg_feature_model.add(Dense(256, activation="relu"))
vgg_feature_model.add(Dropout(0.5))
vgg_feature_model.add(Dense(num_classes, activation="softmax"))

vgg_feature_model.summary()
```

```
Downloading data from https://github.com/rcmalli/keras-vggface/releases/download/v2.0/rcmalli_vggface_tf_notop_vgg16.h5
58916864/58909280 [==============================] - 2s 0us/step
Model: "sequential"
```

Layer (type)	Output Shape	Param #
vggface_vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 256)	131,328
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 7)	1,799

```
Total params: 14,847,815 (56.64 MB)
Trainable params: 133,127 (520.03 KB)
Non-trainable params: 14,714,688 (56.13 MB)
```

Configuración de VGGFace (Fine Tuning)

Para el ajuste fino, se descongeló el bloque 5 del modelo base para permitir que los pesos se ajustaran específicamente al dataset FER-2013.

VGGFace - Ajuste fino

Descongelar solo block5

```
vggface_finetune_model = load_model('/content/drive/MyDrive/FER-2013/modelos/vgg_feature_model.keras')

# En un modelo Sequential, el VGGFace está en la primera capa:
vggface_base = vggface_finetune_model.layers[0]

vggface_base.trainable = True

for layer in vggface_base.layers:
    if 'block5' not in layer.name:
        layer.trainable = False
```

Entrenamiento

```
[ ]
history_vggface_finetune = vggface_finetune_model.fit(
    train_generator_vgg,      # RGB
    validation_data=val_generator_vgg,
    epochs=EPOCHS,
    callbacks=[early_stopping]
)

vggface_finetune_model.save('/content/drive/MyDrive/FER-2013/modelos/vggface_finetune_model.keras')
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `Py
  self._warn_if_super_not_called()
Epoch 1/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 414s 963ms/step - accuracy: 0.6189 - loss: 1.0232 - val_accuracy: 0.6266 - val_loss: 1.0220
Epoch 2/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 348s 861ms/step - accuracy: 0.6257 - loss: 1.0062 - val_accuracy: 0.6259 - val_loss: 1.0224
Epoch 3/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 347s 860ms/step - accuracy: 0.6289 - loss: 0.9952 - val_accuracy: 0.6283 - val_loss: 1.0214
Epoch 4/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 346s 855ms/step - accuracy: 0.6279 - loss: 1.0063 - val_accuracy: 0.6311 - val_loss: 1.0220
Epoch 5/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 345s 853ms/step - accuracy: 0.6272 - loss: 0.9967 - val_accuracy: 0.6290 - val_loss: 1.0222
Epoch 6/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 345s 855ms/step - accuracy: 0.6167 - loss: 1.0222 - val_accuracy: 0.6301 - val_loss: 1.0224
Epoch 7/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 345s 855ms/step - accuracy: 0.6215 - loss: 1.0047 - val_accuracy: 0.6297 - val_loss: 1.0238
Epoch 8/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 347s 859ms/step - accuracy: 0.6277 - loss: 0.9999 - val_accuracy: 0.6273 - val_loss: 1.0227
Epoch 9/50
404/404 ━━━━━━━━━━━━━━━━━━━━ 344s 851ms/step - accuracy: 0.6313 - loss: 0.9964 - val_accuracy: 0.6315 - val_loss: 1.0228
```

Resultados obtenidos

Tras el entrenamiento de los tres modelos y la evaluación sobre el conjunto de test, se obtuvieron las siguientes métricas utilizando el promedio macro para compensar el desbalanceo de clases.

```
[ ]
accuracy_cnn, precision_cnn, recall_cnn, f1_cnn = evaluar_modelo(cnn_model, test_generator)
accuracy_feat, precision_feat, recall_feat, f1_feat = evaluar_modelo(vgg_feature_model, test_generator_vgg)
accuracy_ft, precision_ft, recall_ft, f1_ft = evaluar_modelo(vggface_finetune_model, test_generator_vgg)

print(f"CNN: Accuracy={accuracy_cnn:.4f}, Precision={precision_cnn:.4f}, Recall={recall_cnn:.4f}, F1-score={f1_cnn:.4f}")
print(f"VGG Feature: Accuracy={accuracy_feat:.4f}, Precision={precision_feat:.4f}, Recall={recall_feat:.4f}, F1-score={f1_feat:.4f}")
print(f"VGG Fine-Tune: Accuracy={accuracy_ft:.4f}, Precision={precision_ft:.4f}, Recall={recall_ft:.4f}, F1-score={f1_ft:.4f}")
```

```
225/225 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` clas
  self._warn_if_super_not_called()
113/113 ━━━━━━━━━━━━━━━━━━━━ 40s 351ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 35s 299ms/step
CNN: Accuracy=0.5391, Precision=0.5204, Recall=0.4806, F1-score=0.4878
VGG Feature: Accuracy=0.6226, Precision=0.6256, Recall=0.5436, F1-score=0.5495
VGG Fine-Tune: Accuracy=0.6226, Precision=0.6208, Recall=0.5473, F1-score=0.5554
```

Métrica	CNN Simple	VGGFace (extracción de características)	VGGFace (Fine Tuning)
Accuracy	0,5391	0,6226	0,6226
Precision	0,5204	0,6256	0,6208
Recall	0,4806	0,5436	0,5473
F1-Score	0,4878	0,5495	0,5554

Análisis y comparación de resultados

CNN Simple vs. Transfer Learning

Existe una diferencia muy evidente entre entrenar una red desde cero y utilizar Transfer Learning.

- La CNN Simple obtuvo un accuracy del **53,91%**. Dado que las imágenes originales son de baja resolución (48x48) y en escala de grises, una red poco profunda tiene dificultades para extraer características complejas y robustas de las emociones.
- VGGFace (en ambas variantes) superó el **62%** de accuracy. Esto demuestra la potencia del **Transfer Learning**. VGGFace ha sido entrenada con millones de imágenes de rostros, por lo que sus filtros convolucionales ya saben reconocer formas de ojos, bocas y estructuras faciales.

Extracción de características vs. ajuste fino (Fine Tuning)

Al comparar las dos estrategias de VGGFace, observamos dos detalles:

- **Accuracy idéntico:** Ambos modelos alcanzaron un accuracy de *0,6226*.
- **Mejora en F1-Score:** El modelo con **Fine Tuning** obtuvo un f1-score ligeramente superior (*0,5554* frente a *0,5495*).

El hecho de que el *Fine Tuning* no haya aumentado el accuracy (respecto a la extracción de características) puede deberse a que las imágenes del dataset FER-2013 son de muy baja calidad.

Sin embargo, el aumento en el **f1-score** y el **recall** indica que el ajuste fino ayudó al modelo a balancear mejor las clases minoritarias.

Conclusión general

El uso de modelos pre-entrenados específicos (como VGGFace) es superior a entrenar redes neuronales desde cero, especialmente cuando el dataset es ruidoso o tiene poca resolución como FER-2013. Aunque el ajuste fino (Fine Tuning) es computacionalmente más costoso, ofrece una ligera mejora en la robustez del modelo (f1-score). Esto es algo clave ya que, al tratar con un dataset desbalanceado, priorizar el f1-score frente al accuracy es fundamental para asegurar un rendimiento equilibrado en la detección de todas las clases.